**DISEÑO DE PRUEBAS**

**Clase Bonificación**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que la imagen de la bonificación sea la indicada dependiendo de su tipo. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Bonificacion | BonoVida() | Se crea un BonoVida | Ninguno | Su imagen es igual a la constante imagen de la clase BonoVida |
| Bonificacion | BonoPuntos() | Se crea un BonoPuntos | Ninguno | Su imagen es igual a la constante imagen de la clase BonoPuntos |
| Bonificacion | BonoProyFuerte() | Se crea un BonoProyFuerte | Ninguno | Su imagen es igual a la constante imagen de la clase BonoProyFuerte |
| Bonificacion | BonoProyRapido() | Se crea un BonoProyRapido | Ninguno | Su imagen es igual a la constante imagen de la clase BonoProyRapido |
| Bonificacion | BonoProyNormal() | Se crea un BonoProyNormal | Ninguno | Su imagen es igual a la constante imagen de la clase BonoProyNormal |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que los métodos de Bonificacion getSiguiente, setSiguiente, getAnterior y setAnterior funcionan correctamente al añadir una bonificación a la lista. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | crearBonus(): void | primerBonus=null | Ninguna | Primer bonus es diferente de null. Su siguiente es igual a null |
| Juego | crearBonus(): void | primerBonus creado | Ninguna | El siguiente del primerBonus es diferente de null. El anterior del nuevo añadido es el primerBonus. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método colisionaCon hace invisible al Bonus solo si recibe un objeto Colisionable | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Bonificacion | colsionaCon(Colisionable): void | Una bonificación vida ubicada en (50,50) | Una nave | La bonificación es invisible |
| Bonificacion | colisionaCon(Colisionable): void | Una bonificación vida ubicada en (50,50) | Un proyectil normal | La bonificación es invisible |
| Bonificacion | colisionaCon(Colisionable): void | Una bonificación vida ubicada en (50,50) | Una pelota en nivel 0 | La bonificación es invisible |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método hayColision retorna correctamente si existe colisión. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Bonificación | hayColision(Colisionable): boolean | Una bonificación vida ubicada en (50,50) | Una nave ubicada en (50,50) | True |
| Bonificación | hayColision(Colisionable): boolean | Una bonificación vida ubicada en (50,50) | Una nave ubicada en (100,50) | False |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que se genera correctamente un ArrayList con las bonificaciones de la lista | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | getBonus(): ArrayList<Bonificacion> | La lista de bonificaciones tiene 2 bonificaciones | Ninguno | El arrayList es de tamaño 2. La segunda posición es la siguiente de la primera. La primera es la anterior de la segunda. |
| Juego | getBonus(): ArrayList<Bonificacion> | Un juego con primerBonus = null | Ninguno | El arrayList está vacío |

**Clase Decoracion**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que las bonificaciones son añadidas de la manera correcta | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | agregarDecoracion(Decoracion): void | primeraDeco=null | Una nueva decoración | La primeraDeco es igual a la nueva decoración |
| Juego | agregarDecoracion(Decoracion): void | primeraDeco es una Decoracion | Una nueva decoración | El siguiente de la primeraDeco es igual a la nueva decoración |
| Juego | agregarDecoracion(Decoracion): void | La lista tiene dos decoraciones | Una nueva decoración | El siguiente de la nueva decoración es la primeraDeco, y su anterior es la segunda decoración. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que se genera correctamente un ArrayList con las decoraciones de la lista | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | darDecoraciones(): ArrayList<Decoracion> | La lista de decoraciones tiene 3 decoraciones | Ninguno | El arrayList es de tamaño 3, la primera posición es la primeraDeco, la segunda es el siguiente de la primeraDeco y la tercera posición es la anterior de la primeraDeco |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método crearDecoraciones crea el numero de decoraciones correspondientes a la constante NUMERO\_DECORACIONES | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | crearDecoraciones(): void | Existe un juego. | Ninguno | El numero de decoraciones de la lista es igual a la constante NUMERO\_DECORACIONES |

**Clase Nave**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de colisionaCon realiza su respectiva acción dependiendo del objeto Colisionable recibido | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Nave | colisionaCon(Colisionable): void | Una nave con  X=30  dX=50 | Una pelota en nivel 0 | Su vida es 3, y es invulnerable |
| Nave | colisionaCon(Colisionable): void | Una nave con  X=30  dX=50 | Un bonoVida | Su vida es 5 |
| Nave | colisionaCon(Colisionable): void | Una nave con  X=30  dX=50 | Un bonoProyFuerte | Su proyectil es una instancia de ProyectilFuerte |
| Nave | colisionaCon(Colisionable): void | Una nave con  X=30  dX=50 | Un bonoProyNormal | Su proyectil es una instancia de ProyectilNormal |
| Nave | colisionaCon(Colisionable): void | Una nave con  X=30  dX=50 | Un bonoProyRapido | Su proyectil es una instancia de ProyectilRapido |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método mover cambia de posición correctamente a la nave teniendo en cuenta los límites establecidos. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Nave | mover():void | Una nave con  X=30  dX=50 | Ninguno | X=80 |
| Nave | mover():void | Una nave con  X=810 | Ninguno | X=800-el ancho de la nave |
| Nave | mover():void | Una nave con  X = 50  Y=50  dY=-100 | Ninguno | Y=0 |
| Nave | mover():void | Una nave con  Y=610 | Ninguno | Y=600-el alto de la nave |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método hayColision verifica correctamente si existe una colisión entre la nave y un objeto colisionable. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Nave | hayColision(Colisionable): boolean | Una nave con  X = 50  Y=50  dY=-100 | Una pelota de nivel 0 que está en (50,50) | True |
| Nave | hayColision(Colisionable): boolean | Una nave con  X = 50  Y=50  dY=-100 | Una pelota de nivel 0 que está en (200,200) | False |

**Clase Proyectil**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que la imagen, el daño y la velocidad del proyectil sea la indicada dependiendo de su tipo. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Proyectil | ProyectilFuerte() | Se crea un proyectil fuerte | Ninguno | Su velocidad es 15  Su daño es 5  Su imagen es igual a la constante imagen de la clase ProyectilFuerte |
| Proyectil | ProyectilNormal() | Se crea un proyectil normal | Ninguno | Su velocidad es 20  Su daño es 3  Su imagen es igual a la constante imagen de la clase ProyectilNormal |
| Proyectil | ProyectilRapido() | Se crea un proyectil rápido | Ninguno | Su velocidad es 25  Su daño es 2  Su imagen es igual a la constante imagen de la clase ProyectilRapido |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que la imagen, el daño y la velocidad del proyectil sea la indicada dependiendo de su tipo. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal visible | X=50  Y=50  X2=100  Y2=100 | dX=10  dY=10 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal visible | X=100  Y=100  X2=50  Y2=50 | dX=-23  dY=-23 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal visible | X=100  Y=50  X2=100  Y2=100 | dX=0  dY=13 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal visible | X=100  Y=100  X2=100  Y2=50 | dX=0  dY=-27 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil rápido visible | X=50  Y=50  X2=100  Y2=100 | dX=12  dY=12 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil rápido visible | X=100  Y=100  X2=50  Y2=50 | dX=-29  dY=-29 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil rápido visible | X=100  Y=50  X2=100  Y2=100 | dX=0  dY=17 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil rápido visible | X=100  Y=100  X2=100  Y2=50 | dX=0  dY=-34 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil fuerte visible | X=50  Y=50  X2=100  Y2=100 | dX=7  dY=7 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil fuerte visible | X=100  Y=100  X2=50  Y2=50 | dX=-17  dY=-17 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil fuerte visible | X=100  Y=50  X2=100  Y2=100 | dX=0  dY=10 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil fuerte visible | X=100  Y=100  X2=100  Y2=50 | dX=0  dY=-20 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método mover cambia de posición correctamente del proyectil teniendo en cuenta los límites establecidos. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Proyectil | mover():void | Una Proyectil normal con  X=-50  Y=10 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil normal con  X=10  Y=-50 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil normal con  X=810  Y=10 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil normal con  X=30  Y=610 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil normal con  X=50  Y=50 | Ninguno | Es visible |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de colisionaCon realiza su respectiva acción dependiendo del objeto Colisionable recibido | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Proyectil | colisionaCon(Colisionable): void | Hay un proyectil normal | Una pelota | El proyectil es invisible |
| Proyectil | colisionaCon(Colisionable): void | Hay un proyectil normal | Un bonoVida | El proyectil es visible |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método hayColision verifica correctamente si existe una colision entre el proyectil y un objeto colisionable. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Proyectil | hayColision(Colisionable): boolean | El proyectil en  (50,50) | Una pelota que está en (50,50) | True |
| Proyectil | hayColision(Colisionable): boolean | El proyectil en  (50,50) | Una pelota que está en (200,200) | False |

**Clase Pelota**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de hayColision retorna efectivamente si existe o no Colision | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | hayColision(Colisionable): boolean | Hay una pelota con ubicación (50.,50) | Un Proyectil con x=50 y=50 | True |
| Pelota | hayColision(Colisionable):  boolean | Hay una pelota con ubicación (50.,50) | Un Proyectil con x=100 y=100 | False |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de colisionaCon realiza su respectiva acción dependiendo del objeto Colisionable recibido | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | colisionaCon(Colisionable): void | Hay una pelota | Un Proyectil Normal | Vida = 2 |
| Pelota | colisionaCon(Colisionable): void | Hay una pelota | Un Proyectil Rápido | Vida = 3 |
| Pelota | colisionaCon(Colisionable): void | Hay una pelota | Un Proyectil Fuerte | Vida = 0  Visible = false |
| Pelota | colisionaCon(Colisionable): void | Hay una pelota | Una nave | Vida = 5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de existenColisiones verifica exitosamente si existe alguna colisión | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | existenColisiones(Colsionable): boolean | El árbol de pelotas tiene 3 pelotas ubicadas en  Pelota 1: (50,50)  Pelota 2: (100,100)  Pelota 3: (40,100) | Un Proyectil Normal ubicado en (50,50) | true |
| Pelota | existenColisiones(Colsionable): boolean | El árbol de pelotas tiene 3 pelotas ubicadas en  Pelota 1: (50,50)  Pelota 2: (100,100)  Pelota 3: (40,100) | Un Proyectil Normal ubicado en (100,100) | true |
| Pelota | existenColisiones(Colsionable): boolean | El árbol de pelotas tiene 3 pelotas ubicadas en  Pelota 1: (50,50)  Pelota 2: (100,100)  Pelota 3: (40,100) | Un Proyectil Normal ubicado en (40,100) | true |
| Pelota | existenColisiones(Colsionable): boolean | El árbol de pelotas tiene 3 pelotas ubicadas en  Pelota 1: (50,50)  Pelota 2: (100,100)  Pelota 3: (40,100) | Un proyectil normal ubicado en (10,20) | false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de hayVivas verifica exitosamente si existe alguna Pelota viva | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | hayVivas(): boolean | El árbol de pelotas tiene 3 pelotas  Pelota 1: visible  Pelota 2: visible  Pelota 3: visible | Ninguno | true |
| Pelota | hayVivas(): boolean | El árbol de pelotas tiene 3 pelotas  Pelota 1: visible  Pelota 2: invisible  Pelota 3: visible | Ninguno | true |
| Pelota | hayVivas(): boolean | El árbol de pelotas tiene 3 pelotas  Pelota 1: visible  Pelota 2: visible  Pelota 3: invisible | Ninguno | true |
| Juego | hayVivas(): boolean | El árbol de pelotas tiene 3 pelotas  Pelota 1: invisible  Pelota 2: invisible  Pelota 3: invisible | Ninguno | false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de disminuirVida disminuye exitosamente la vida de la pelota de acuerdo con el Proyectil. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | disminuirVida(Proyectil): void | Una pelota | Un proyectil normal | Su vida es 2 |
| Pelota | disminuirVida(Proyectil): void | Una pelota | Un proyectil rapido | Su vida es 3 |
| Pelota | disminuirVida(Proyectil): void | Una pelota | Un proyectil fuerte | Su vida es 0, y es invisible. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de insertar añade exitosamente una pelota al árbol. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | insertarPelota(Pelota):void | raizPelota=null | Una pelota | La raíz del árbol es esa pelota insertada. |
| Juego | insertarPelota(Pelota):void | raizPelota=Pelota | Una pelota | Tiene subárbol izq o der, su peso es igual a 2. |
| Juego | insertarPelota(Pelota):void | Un árbol con dos pelotas | Una pelota | El peso del árbol es 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que se genera correctamente un ArrayList con las pelotas del árbol | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | getPelotas(): ArrayList<Pelota> | El árbol de pelotas tiene 3 pelotas ubicadas en  Pelota 1: (50,50)  Pelota 2: (100,100)  Pelota 3: (40,100) | Ninguno | El arrayList es de tamaño 3, la segunda posición del arreglo es la raizPelotas. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método mover cambia de posición correctamente a la pelota teniendo en cuenta los límites establecidos. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Pelota | mover():void | Una pelota con  X=10  Y=10  dX=10  dY=10 | Ninguno | X=20  Y=20 |
| Pelota | mover():void | Una pelota con  X=10  Y=10  dX=-20  dY=-20 | Ninguno | dX=20  dY=20 |
| Pelota | mover():void | Una pelota con  X=10  Y=0  dX=0  dY=-10 | Ninguno | dY=10 |
| Pelota | mover():void | Una pelota con  X=800  Y=10  dX=10  dY=10 | Ninguno | dX=-10 |

**Clase Juego**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de iniciarJuego inicia el juego de manera correcta | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | iniciarJuego(boolean cargado): void | Un juego recién creado  Jugador  Nickname: Armando | False | Crea una nave nueva  El número de pelotas es 3  Hay tres pelotas en el árbol  jugando= true  La cantidad de decoraciones es 5 |
| Juego | iniciarJuego(boolean cargado): void | Un juego en nivel 4  Jugador Nickname:  Armando | False | Crea una nave nueva  El número de pelotas es 4  Hay cuatro pelotas en el árbol  jugando= true |
| Juego | iniciarJuego(boolean cargado): void | Inicia un juego ya existente:  Juego:  Nickname Jugador = “Armando”  Nivel=2  Puntos=20  Nave= Tiene 2 vidas y un proyectilFuerte | True | El nickname del jugador es Armando, su nivel es 2, hay 3 pelotas en la pantalla, tiene 20 puntos, la nave tiene dos vidas y un proyectil fuerte. |
| Juego | iniciarJuego(boolean cargado): void | En el juego existe un jugador de nickname Camila  Se crea un nuevo juego y  Jugador nickname: Camila | False | Lanza Excepcion JugadorRepetidoException |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de subirNivel crea el número de Pelotas correspondientes al nivel alcanzado | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | subirNivel() | Crea un juego  Llama al método iniciarJuego(false) | Ninguno | El peso del árbol pelotas es 3 |
| Juego | subirNivel() | Crea un juego  Llama al método iniciarJuego(false)  Se llama al método subir nivel tres veces (se encuentra en nivel 4) | Ninguno | El peso del árbol pelotas es 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de bonusPuntaje incremente el puntaje en 10 puntos | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | bonusPuntaje() | Un juego con 10 puntos | Ninguno | El puntaje es 20 |
| Juego | bonusPuntaje() | Un juego con 20 puntos | Ninguno | El puntaje es 30 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de aumentarPuntaje incremente el puntaje en 5 puntos | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | bonusPuntaje() | Un juego con 10 puntos | Ninguno | El puntaje es 15 |
| Juego | bonusPuntaje() | Un juego con 5 puntos | Ninguno | El puntaje es 10 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que los métodos se serialización y recuperado de los elementos del juego funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | recuperarNave(String):  void | Una nave con 2 vidas y un proyectil rápido.  Se llama al método guardarNave(String) donde  String direc= “./test/datatest/nave.txt” | “./test/datatest/nave.txt” | La nave recuperada tiene 2 vidas y un proyectil rápido |
| Juego | recuperarNave(String): void | El mismo que el anterior | “./datatest/nave.txt” | Lanza una IOException |
| Juego | recuperarPelotas(String): void | Un juego con un árbol de pelotas de peso 4.  Se llama al método  guardarPelotas(String) donde  String direc= “./test/datatest/pelotas.txt” | “./test/datatest/pelotas.txt” | El árbol de pelotas es de peso 4 |
| Juego | recuperarPelotas(String): void | El mismo que el anterior | “./datatest/pelotas.txt” | Lanza IOException |
| Juego | recuperarBonus(String): void | Un juego con una lista de bonusque tenga 4 bonus.  Se llama al método  guardarBonus(String) donde  String direc= “./test/datatest/bonus.txt” | “./test/datatest/bonus.txt” | La lista de bonos es de tamaño 4. |
| Juego | recuperarBonus(String): void | El mismo que el anterior | “./datatest/bonus.txt” | Lanza IOException |
| Juego | recuperarDeco(String): void | Un juego con una lista de 3 decoraciones  Se llama al método  guardarDeco(String) donde  String direc= “./test/datatest/deco.txt” | “./test/datatest/deco.txt” | La lista recuperada tiene 3 decoraciones. |
| Juego | recuperarDeco(String): void | El mismo que el anterior | “./datatest/deco.txt” | Lanza IOException |
| Juego | recuperarJugadores(String): void | Un juego con un árbol de jugadores de peso 5, donde su raíz es un jugador con  Nickname=”Joan”  Puntaje=100  Nivel=1  Se llama al método  guardarJugadores(String) donde  String direc= “./test/datatest/users.txt” | “./test/datatest/users.txt” | El árbol recuperado es de peso 5 y su raíz es un jugador de nickname Joan, puntaje 10 y nivel 1. |
| Juego | recuperarJugadores(String): void | El mismo que el anterior | “./datatest/users.txt” | Lanza IOException |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que los métodos de manejo de archivos de texto del juego funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | cargarDatos(String):  void | Un juego con  Puntaje=15  Nivel=3  Y un jugador con nickname=”Javier”  Se llama al método guardarDatos(String) donde  String direc= “./test/datatest/data.txt” | “./test/datatest/data.txt” | El juego tiene  Puntaje=100  Nivel=10  Y un jugador con nickname=”Javier” |
| Juego | cargarDatos(String):  void | El mismo que el anterior | “./datatest/data.txt” | Lanza una IOException |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método verificarVidas para o no el juego dependiendo si la nave sigue viva o no | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | verificarVidas():void | Un juego con una nave recién creada | Ninguna | Jugando=true |
| Juego | verificarVidas():void | Un juego con una nave sin vidas (-1 vidas) | Ninguna | Jugando=false |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método verificarColisionNave vuelve invulnerable la nave si alguna de las pelotas de la pantalla ha colisionado con ella | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | verificarColisionNave(): void | Un juego con una nave recién creada en (50,50).  El árbol de pelotas tiene dos pelotas  Pelota 1 está ubicada en (200,200) y pelota 2 está ubicada en (50,50) | Ninguna | La nave es invulnerable y su vida es igual a 3 |
| Juego | verificarColisionNave(): void | Un juego con una nave recién creada en (50,50).  El árbol de pelotas tiene dos pelotas  Pelota 1 está ubicada en (200,200) y pelota 2 está ubicada en (300,300) | Ninguna | La nave no es invulnerable y su vida es igual a 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método verificarColisionBonus realiza la acción correspondiende si alguna de las bonificaciones de la lista ha colisionado con la nave | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | verificarColisionBonus(): void | Un juego con una nave recién creada en (50,50).  Una lista de bonificaciones donde la primera bonificación está ubicada en (50,50)  El juego tiene puntaje 10 | Ninguna | Alguno de los siguientes efectos, dependiendo del tipo bonus que se haya generado:   * Si es BonusPuntos, el puntaje del juego es 20 * Si es BonusVida, la nave tiene 5 vidas * Si es BonusProyNormal, el proyectil de la nave es un proyectil normal * Si es BonusProyFuerte, el proyectil de la nave es un proyectil fuerte * Si es BonusProyRapido, el proyectil de la nave es un proyectil rapido |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método verificarColisionProyectil disminuye la vida de las pelotas o aumenta el puntaje si se queda sin vidas | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | verificarColisionProyectil(): void | Un juego con un proyectil fuerte visible ubicado en (50,50) y un árbol de pelotas con una raíz que tiene  Vida=5  Ubicación=50,50  El puntaje del juego es 10 | Ninguna | La raíz del árbol de pelotas es invisible  El puntaje es 15 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de addJugador añade correctamente un Jugador | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | addJugador(): void | Raizjugador = null  Jugador =  Nickname: “Joan”  Puntos: 100  Nivel: 1 | Ninguna | Raizjugador es igual el jugador |
| Juego | addJugador(): void | Raizjugador= Joan  Jugador =  Nickname: “Camila”  Puntos: 50  Nivel: 1 | Ninguna | Camila es el subárbol izquierdo de Joan |
| Juego | addJugador(): void | Raizjugador= Joan  Jugador =  Nickname: “Sariana”  Puntos: 500  Nivel: 4 | Ninguna | Sariana es el subárbol derecho de Joan |
| Juego | addJuagdor(): void | Raizjugador = Joan  Jugador =  Nickname: “Joan”  Puntos: 200  Nivel: 1 | Ninguna | Lanza JugadorRepetidoException |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de mostrar la lista ordenada por puntaje de manera ascendente y descendente funciona bien. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | ordernarPuntajeAscendente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Alejandro  Julian  Javier  JuanMa  Manyolml |
| Juego | ordernarPuntajeDescendente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Manyolml  JuanMa  Javier  Julian  Alejandro |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de mostrar la lista ordenada por nombre de manera ascendente y descendente funciona bien. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | ordernarNombreAscencente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Manyolml  Julian  JuanMa  Javier  Alejandro |
| Juego | ordernarNombreDescencente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Alejandro  Javier  JuanMa  Julian  Manyolml |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de mostrar la lista ordenada por nivel de manera ascendente y descendente funciona bien. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | ordernarNivelAscencente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Alejandro  Julian  Javier  JuanMa  Manyolml |
| Juego | ordernarNivelAscencente(): ArrayList<Jugador> | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | Ninguna | Manyolml  JuanMa  Javier  Julian  Alejandro |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de buscar un jugador por puntaje encuentra al jugador correspondiente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | buscarJugadorPuntos(int puntos): Jugador | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | 500 | Retorna a Manyo |
| Juego | buscarJugadorPuntos(int puntos): Jugador | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | 2 | Lanza PuntajeNoExisteException |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prueba: Verifica que el método de buscar un jugador por puntaje encuentra al jugador correspondiente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Juego | BuscarJugadorNombre(String nombre): Jugador | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | “Javier” | Retorna al jugador “Javier” |
| Juego | buscarJugadorNombre(String nombre): Jugador | Hay un árbol binario donde la raíz es  Nickname: “Julian”  Puntos: 100  Nivel: 3  Y en conjunto el árbol tiene los siguientes elementos  Nickname: “Javier”  Puntos: 200  Nivel: 4  Nickname: “Alejandro”  Puntos: 50  Nivel: 2  Nickname: “Manyolml”  Puntos: 500  Nivel: 7  Nickname: “JuanMa”  Puntos: 400  Nivel: 6 | “Karol” | Lanza NombreNoExisteException |