**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 | BonoProyFuerte | Crea un BonoProyFuerte  Bono bono = BonoProyFuerte() | Ninguno | La imagen de bono es igual a la constante DIREC de la clase BonoProyFuerte |
| Bonificacion | BonoProyNormal | Crea un BonoProyNormal | Ninguno | La imagen de bono es igual a la constante DIREC de la clase BonoProyNormal |
| Bonificacion | BonoProyRapido | Crea un BonoProyRapido | Ninguno | La imagen de bono es igual a la constante DIREC de la clase BonoProyRapido |
| Bonificación | BonoVida | Crea un BonoVida | Ninguno | La imagen de bono es igual a la constante DIREC de la clase BonoVida |
| Bonificación | BonoPuntos | Crea un BonoPuntos | Ninguno | La imagen de bono es igual a la constante DIREC de la clase BonoPuntos |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | El siguiente del primerBonus es él mismo, y el anterior también. |
| Juego | crearBonus(): void | primerBonus=BonoVida | Ninguna | El siguiente del primerBonus es él nuevo añadido. El siguiente del nuevo añadido es el primerBonus y el anterior también. |
| Juego | crearBonus(): void | primerBonus=BonoVida  El siguiente es un BonoProyFuerte | Ninguna | El siguiente del primerBonus es el BonoProyFuerte y el siguiente del BonoProyFuerte es el nuevo bonus añadido, y el siguiente de este es el primerBonus, y el anterior del nuevo añadido es el BonoProyFuerte. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | Existe una bonificación | Una nave | La bonificación es invisible |
| Bonificacion | colisionaCon(Colisionable): void | Existe una bonificación | Un proyectil | La bonificación es invisible |
| Bonificacion | colisionaCon(Colisionable): void | Existe una bonificación | Una pelota | 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 | La bonificación está ubicada en (50,50) | Una nave ubicada en (50,50) | True |
| Bonificación | hayColision(Colisionable): boolean | La bonificación está 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 3 bonificaciones | Ninguno | El arrayList es de tamaño 3, la primera posición es el primerBonus, la segunda es el siguiente del primerBonus y la tercera posición es la anterior del primerBonus |

**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 | 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 |

**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 | Se crea una nave. | Una pelota | Su vida es 3, y es invulnerable |
| Nave | colisionaCon(Colisionable): void | Se crea una nave. | Un bonoVida | Su vida es 5 |
| Nave | colisionaCon(Colisionable): void | Se crea una nave. | Un bonoProyFuerte | Su proyectil es una instancia de BonoProyFuerte |
| Nave | colisionaCon(Colisionable): void | Se crea una nave. | Un bonoProyNormal | Su proyectil es una instancia de BonoProyNormal |
| Nave | colisionaCon(Colisionable): void | Se crea una nave. | Un bonoProyNormal | Su proyectil es una instancia de BonoProyRapido |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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=0 |
| Nave | mover():void | Una nave con  X=810 | Ninguno | X=800-el ancho de la nave |
| Nave | mover():void | Una nave con  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 colision entre la nave y un objeto colisionable. | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Nave | hayColision(Colisionable): boolean | La nave esta en  (50,50) | Una pelota que está en (50,50) | True |
| Nave | hayColision(Colisionable): boolean | La nave está en  (50,50) | Una pelota 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 | X=50  Y=50  X2=100  Y2=100 | dX=14  dY=14 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal | X=100  Y=100  X2=50  Y2=50 | dX=-14  dY=-14 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal | X=100  Y=50  X2=100  Y2=100 | dX=0  dY=20 |
| Proyectil | Disparar(int, int, int, int):void | Un proyectil normal | 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 con  X=-50  Y=10 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil con  X=10  Y=-50 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil con  X=810  Y=10 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil con  X=30  Y=610 | Ninguno | Dx=0  Dy=0  Es invisible |
| Proyectil | mover():void | Una Proyectil 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 | Una bonificación | 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 mismo anterior | Un Proyectil Normal ubicado en (100,100) | true |
| Pelota | existenColisiones(Colsionable): boolean | El mismo anterior | Un Proyectil Normal ubicado en (40,100) | true |
| Pelota | existenColisiones(Colsionable): boolean | El mismo anterior | 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 tres pelotas | 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 2 nuevo  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 | True | La nave es la nave ya existente  El nickname del jugador es el ya existente  El puntaje del jugador es el ya existente  El nivel del jugador es el ya existente |
| Juego | iniciarJuego(boolean cargado): void | Inicia un juego nuevo con un jugador ya insertado  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 niveldos veces (se encuentra en nivel 3) | Ninguno | El peso del árbol pelotas es 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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> | El mismo de arriba | 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> | El mismo de arriba | Ninguna | Manyolml  Julian  JuanMa  Javier  Alejandro |
| Juego | ordernarNombreDescencente(): ArrayList<Jugador> | El mismo de arriba | 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> | El mismo de arriba | Ninguna | Alejandro  Julian  Javier  JuanMa  Manyolml |
| Juego | ordernarNivelAscencente(): ArrayList<Jugador> | El mismo de arriba | 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 | El mismo de arriba | 500 | Retorna a Manyo |
| Juego | buscarJugadorPuntos(int puntos): Jugador | El mismo de arriba | 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 | El mismo de arriba | “Javier” | Retorna al jugador “Javier” |
| Juego | buscarJugadorNombre(String nombre): Jugador | El mismo de arriba | “Karol” | Lanza NombreNoExisteException |