

Matías Eric Viglione Muñoz (1423588)
Mohamed Yassin El Hajouji El Hayadi (1421980)
Snake

Cell

Funcionalidad: Constructor de la clase Cell, según el tipo de CellType el objeto se le asignara un score de manera que EMPTY, SMALL_ITEM, MED_ITEM, BIG_ITEM, WALL y SNAKE tendrán respectivamente 10, 30, 50, 100, 0 y 0 de score.

Localización: Cell.java, Cell(CellType cellType)

Test: CellTest.java, testCell().

Test de caja negra: partición equivalente.

Donde probamos las 5 particiones de inputs que acepta el constructor, WALL y SNAKE serian del mismo grupo al dar el mismo score y los demás supondrían un grupo cada uno.

Funcionalidad: Modifica el objeto Cell según el tipo de CellType dado, no solo modifica el cellType si no que el score debe coincidir siguiendo las normas del constructor

Localización: Cell.java, setCellType(CellType cellType)

Test: CellTest.java, testSetCellType().

Test de caja negra: partición equivalente.

Probamos como en el test del constructor las 5 particiones de inputs que acepta la función, WALL y SNAKE serian del mismo grupo al dar el mismo score y los demás supondrían un grupo cada uno.

Funcionalidad: Modifica el objeto Cell según el score introducido, hay 5 tipos de score para, introducir un score fuera de estos será lo mismo que tener una celda de tipo EMPTY con 10 de score y si se introduce 0 pondrá como tipo de celda WALL con score 0.

Localización: Cell.java, setScore(int score)

Test: CellTest.java, testSetScore().

Test de caja negra: partición equivalente, valores límites y frontera.

Probamos los valores limite y frontera de cada partición, para cada score probaremos el score mismo, un valor entre 2 scores y el valor límite inferior y superior de cada score.

Scores	10	30	50	100	0
test values	9, 10, 11	29, 30, 31	49, 50, 51	99, 100, 101	-1, 0, 1

valores interiores a frontera: 5, 20, 40, 80

valores exteriores a frontera: -50, 150

Funcionalidad: devuelve el valor score del objeto Cell

Localización: Cell.java, getScore()

Test: CellTest.java, testGetScore().

Test de caja negra: partición equivalente.

Comprobamos el score del getter para cada celda inicializada con una de las 5 particiones.

Funcionalidad: devuelve el valor de CellType del objeto Cell

Localización: Cell.java, getCellType()

Test: CellTest.java, testGetCellType().

Test de caja negra.

Comprobamos el CellType devuelto por getter para cada celda inicializada con una de los 6 tipos de celda.

Snake

Funcionalidad: Constructor de la clase Snake. El constructor debe comprobar que los valores x, y a guardar sean ambos positivos y en caso contrario guardará 0,0.

Localización: Snake.java, Snake(int x, int y)

Test: SnakeTest.java, testSnake()

Test de caja negra: pairwise pairing

Comprobar combinaciones entre poner un negativo en la x o en la y

casos a probar: (-x, y) , (-x,-y), (x,-y), (x, y)

Test: (-1,1) , (-1,-1), (1,-1), (1,1), (0,0)

Funcionalidad: devuelve el valor deseado del objeto Snake

Localización: Snake.java, getX(), getY()

Test: SnakeTest.java, testGetX(), testGetY().

Test de caja negra.

Comprobamos si el valor devuelto por el getter corresponde al correcto.

Funcionalidad: Modifica las x y las y del objeto snake, debe comprobar que los valores x,y a guardar sean ambos positivos y en caso contrario guardará 0,0.

Localización: Snake.java, setPos(int x, int y)

Test: SnakeTest.java, testSetPos()

Test de caja negra: pairwise pairing

Comprobar combinaciones entre poner un negativo en la x o en la y

casos a probar: (-x, y) , (-x,-y), (x,-y), (x, y)

Test: (-1,1) , (-1,-1), (1,-1), (1,1), (0,0)

Score

Funcionalidad: Constructor de Score que inicializa los valores score y time en 0 y el string userName en "".

Localizacion: Score.java, Score()

Test: ScoreTest, testScore().

Test de caja negra

Comprobamos que la inicialización de las variables es correcta.

Funcionalidad: devuelve el valor deseado del objeto Score

Localización: Score.java, getUsername(), GetUserScore(), getDuration()

Test: ScoreTest, testGetUsername(), testGetUserScore(), testGetDuration().

Test de caja negra

Comprobamos si el valor devuelto por el getter corresponde al correcto.

Funcionalidad: asigna el valor deseado a la variable

Localización: Score.java, setDuration(long duration), setUserScore(int userScore)

Test: ScoreTest, testSetDuration(), testSetUserScore().

Test de caja negra

Comprobamos que el valor asignado es correcto.

Funcionalidad: suma el valor de entrada a la variable

Localización: Score.java, addDuration(long duration), addUserScore(int userScore)

Test: ScoreTest, testAddDuration(), testAddDuration().

Test de caja negra

Comprobamos que el valor es correcto.

Funcionalidad: Modifica la variable name, la variable name solo debe contener número y letras y tener 6 letras o menos, en caso de que tenga más se quedaran con las 6 primeras letras y en caso de que contenga un char que no sea letra o número el nombre será “_”.

Localización: Score.java, setName()

Test: ScoreTest, testSetName().

Test de caja negra: particiones equivalentes y valores límite y frontera.

Comprobaremos valores de string que tengan una longitud menor a 6, igual a 6 y superior a 6. También comprobaremos entrada de solo letras, solo números y números y letras. También comprobaremos string con valores que no son letras ni números como “-”.

Board

Funcionalidad: Constructor de Board. Crea una matriz cells a partir de un número de fila y columna dado, también añade celdas WALL alrededor de la matriz.

Localización: Board.java, Board(int rowSize, int colSize)

Test: BoardTest.java, testBoard().

Test de caja negra

Comprobar que en las 4 esquinas hay celdas tipo WALL, que en el centro hay celda tipo EMPTY.

Funcionalidad: devuelve el valor deseado del objeto Score

Localización: Board.java, getRowSize(), getColSize(), getCell(int x, int y), getHead(), getTail().

Test: BoardTest.java, testGetRowSize(), testGetColSize(), testGetCell(), testGetHead(), testGetTail().

Test de caja negra

Comprobamos si el valor devuelto por el getter corresponde al correcto.

Funcionalidad: Devuelve true si una celda corresponde al tipo BIG_ITEM, SMALL_ITEM o MED_ITEM

Localización: Board.java, isItem(Cell.CellType cell)

Test: BoardTest.java, testIsItem().

Test de caja negra

Comprobamos si el valor devuelto por corresponde al correcto para 6 tipos de celda.

Funcionalidad: asigna el valor deseado a la variable

Localización: Board.java, setHead(int, int), setCells(int, int, Cell.CellType)

Test: BoardTest.java, testSetHead(), testSetCells().

Test de caja negra

Comprobamos que el valor asignado es correcto.

Funcionalidad: Mueve la serpiente. Mueve de casilla a la serpiente según la dirección guardada en la variable direction y guarda la posición en la primera casilla de la lista de snakes

Localización: Board.java, move()

Test: BoardTest.java, testMove()

Test de caja blanca - decision coverage

Comprobar, para las cuatro direcciones posibles (UP, DOWN, LEFT, RIGHT), si después de que se ejecute el movimiento a la casilla destino, dicha casilla contenga la serpiente y que sea de tipo SNAKE. También comprobar si la nueva posición ha sido guardada en la primera posición de la lista de snakes.

Funcionalidad: Si se le introduce false(no ha comido un objeto) elimina de la lista de snake el valor más antiguo y devuelve a EMPTY la posición asociada a este en la tabla de celdas en caso contrario añade una casilla más a la lista de snake.

Localización: Board.java, updateSnake(boolean obj)

Test: BoardTest.java, testUpdateSnake()

Test de caja blanca - decision coverage

Comprobar para true si la cola es ahora más larga y por lo tanto la celda sigue siendo snake después de 2 turnos y para false si la cola ya no está en la última casilla.

Game

Funcionalidad: Observar los límites de una tabla al crear el juego, miramos que cuanto el valor sea demasiado bajo o alto, se le establezcan los valores mínimos y máximos establecidos.

Localización: Game.java, Game(int rowSize, int colSize)

Test: GameTest.java, testGame()

Test de caja negra: partición equivalente, valores límites y frontera.

Primero comprobamos valores interiores, los cuales se guardan por defecto. Luego, comprobamos valores exteriores superiores a los límites, y esperamos que devuelva su valor limite máximo, que es 50. Luego valores exteriores inferiores a los límites, y esperamos que devuelva su valor limite mínimo, que es 10. Y por último comprobamos los valores limites, y observamos si devuelven el mismo valor introducido.

Funcionalidad: Mirar que el proceso Start finalice sin problemas.

Localización: Game.java, start()

Test: GameTest.java, testStart()

Test de caja blanca - Path converge

Ejecutamos el código start y si finaliza nos devuelve un True.

Funcionalidad: Loop del juego, es el loop que se repetirá contantemente hasta que la serpiente se choque con una pared o con ella misma.

Localización: Game.java, play(Board.Direction direction)

Test: GameTest.java, testPlay()

Test de caja blanca – Loop converge (con uso de mocks)

Repetiremos el bucle y veremos si cumple su condición de salida, para eso crearemos una pared justo delante de la serpiente, y la iremos moviendo una casilla mas atrás en cada test, por cada casilla recorrida la parida adquiere 10 puntos, por lo que al final comprobaremos que se hayan recogido los puntos necesarios al finalizar el bucle.

Funcionalidad: Agrega puntos según la casilla recorrida.

Localización: Game.java, addScore(Cell.CellType e)

Test: GameTest.java, testAddScore()

Test de caja blanca – decisión converge

Exploramos todos los tipos diferentes de casillas, y observamos si los puntos adquiridos son los que indicados. EMPTY = 10, SMALL_ITEM = 30, MED_ITEM = 50, BIG_ITEM = 100, SNAKE = 0, WALL = 0.

Funcionalidad: Crea un nuevo objetivo en el tablero en una posición aleatoria que no sea pared ni serpiente.

Localización: Game.java, NewObjective()

Test: GameTest.java, testNewObjective()

Test de caja negra: partición equivalente

Ejecutamos la función una vez, y buscamos en toda la tabla si se encuentra un objeto.

Funcionalidad: Devuelve de forma aleatoria uno de los tres tipos diferentes de Item que hay, BIG_ITEM, MED_ITEM, SMALL_ITEM.

Localización: Game.java, randomItem()

Test: GameTest.java, testRandomItem()

Test de caja negra: partición equivalente

Ejecutamos la función tres veces, y observamos si devuelve un Item.

Funcionalidad: Pantalla de finalización del juego, donde los datos se guardan e introducimos el nombre del usuario

Localización: Game.java, GameOver()

Test: GameTest.java, testGameOver()

Test de caja negra: partición equivalente

Ejecutaremos la función y le guardamos un string, y observamos si se ha guardado.

Ranking

Funcionalidad: Constructor de Ranking, construye una lista con datos sobre la partida a partir de una clase game. Al principio solo contiene una fila con los datos del primer game.

Localización: Ranking.java, Ranking(Game game)

Test: RankingTest.java, testRanking()

Test de caja negra (usando mock object de Game)

Para comprobar el funcionamiento del constructor le hemos dado un mock object de Game que siempre devuelve los mismos valores y que por lo tanto no necesita ser ejecutado.

Funcionalidad: Añade una fila más a la lista de puntuación con los datos de un objeto game.

Localización: Ranking.java, write(Game game)

Test: RankingTest.java, testWrite()

Test de caja negra (usando mock object de Game)

Hemos usado el mock object ya creado para comprobar que añade una fila más a la lista.

MainApp

Funcionalidad: El main muestra el menú y llama a las funciones de game y ranking.

Localización: mainApp.java, main(String[] args)


















Test: mainAppTest.java, testMain()

Test de caja blanca: loop simple (usando mock object de Game)

Para evitar tener que ejecutar la función start() de Game para cada test loop hemos creado un mock object de Game que devuelve valores fijos y tiene una función start() que no hace nada.

Comprobamos que main se ejecuta sin problemas en 4 casos. Sin entrar al bucle, entrando al bucle 1 vez, entrando 2 veces y entrando 3 veces. No podemos comprobarlo en el caso de máximas iteraciones porque este es un bucle que termina cuando el usuario quiere.

Statement coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Snake	 95,2 %	3.400	171	3.571
src	 95,2 %	3.400	171	3.571
(default package)	 95,2 %	3.400	171	3.571
Game.java	 80,2 %	287	71	358
GameTest.java	 96,2 %	608	24	632
CellTest.java	 95,6 %	431	20	451
Board.java	 96,7 %	553	19	572
SnakeTest.java	 95,2 %	240	12	252
ScoreTest.java	 96,4 %	264	10	274
BoardTest.java	 98,2 %	438	8	446
Ranking.java	 96,9 %	123	4	127
mainApp.java	 95,6 %	65	3	68
Cell.java	 100,0 %	131	0	131
mainAppTest.java	 100,0 %	73	0	73
RankingTest.java	 100,0 %	92	0	92
Score.java	 100,0 %	67	0	67
Snake.java	 100,0 %	28	0	28