Grupo 9.3

Justificación de los juegos de prueba:

- Generador:

El testing general de esta funcionalidad está plasmado en los gráficos de "Generador_Performance.pdf", que demuestran la mejoría general del Generador en comparación con su versión anterior. En este documento se comentan los juegos de pruebas más específicos de las NUEVAS funciones internas del Generador. Las antiguas no son mencionadas: ya se comentaron en la primera entrega.
En este fichero, únicamente se adjuntan los juegos de prueba de los casos deterministas, o de aquellos no deterministas pero que tienen algún caso específico en el que sí lo son. Se comentan los no deterministas, indicando qué se espera que hagan, pero sin inputs.No obstante, en el fichero GeneradorDriver, se puede encontrar código para testear más funciones (las de la primera entrega).
El fichero jocs_proves.in contiene los juegos de prueba. En jocs_proves_esperada.out tiene la salida que se espera, para así compararla con la obtenida.
Los casos nuevos son los siguientes (el número de caso es únicamente para la selección, algunos números reemplazan funciones de la primera entrega):

Caso 2: omplir -> No determinista, se encarga de llenar board con valores que cumplen la restricción de no repetición. Es a prueba de errores: devuelve error si el tablero tiene algún fallo.

Caso 3: calc_sum_uniq -> No determinista, tiene la funcionalidad de devolver un vector con una partición única con un número fijo de casillas, pudiendo tener valores algunas de ellas. No es determinista porque suele haber diferentes particiones únicas dado un número de casillas. Y esta función devuelve alguna de ellas, elegida aleatoriamente. Por lo tanto, puede ser determinista en algunos casos (en especial, cuando no encuentra ninguna posible combinación debido a las casillas que ya están completas).

Caso 7: check_zeros -> Determinista, encargada de detectar e intentar arreglar una secuencia de más de 9 ceros seguidos. Es una función nueva, pero no es una funcionalidad nueva. El código estaba antes en generarBoard, y ahora es una función nueva, por motivos de portabilidad y legibilidad del código. No testearemos este caso porque no es código nuevo. Sólo lo mencionamos porque es una función que antes no existía.

Caso 8: omplir_smart2 -> No determinista, se encarga de llenar board con valores que cumplen la restricción de no repetición y que genera una suma con particiones únicas.

Caso 9: omplir_smart -> No determinista, se encarga de llenar board con valores que cumplen la restricción de no repetición. Puede sobrescribir valores escritos por omplir_smart2 para así poder completar la board. Es a prueba de errores: devuelve error si el tablero tiene algún fallo.

Caso 10: sumes_paraleles -> Determinista, se encarga detectar aquellos casos en que hay restricciones de fila o de columna paralelas sin casillas negras de por medio y con el mismo número de casillas blancas y el mismo valor de suma. Es decir, que nunca tendrá solución única porque estas dos restricciones paralelas podrían intercambiarse cualquier valor.

Caso 11: código error omplir_smart -> Determinista. Básicamente, devuelve el código que devuelve la función (0 si va bien, -1 si ha detectado alguna anomalía). Su creación se debe a que así podemos testear, con un output fijo, la funcionalidad de búsqueda de errores de esta función.

Grupo 9.3

Ahora, analizaremos los juegos de pruebas de cada caso:

Caso 2: interesa testear, sobre todo, que deja ciertas casillas blancas con valores (tantas como la variable resolved), que las demás casillas blancas las pone a 0. Interesa también ver que reacciona bien ante los errores que le comunica omplir_smart. Como es una función no determinista, la hemos testeado unas cuantas veces y hemos visto que reacciona como se espera que lo haga.

Caso 3: interesa testear y ver que, efectivamente, devuelve particiones únicas teniendo en cuenta el vector de presencia que se le envía. Adjuntamos dos juegos de pruebas que sí que son deterministas: el caso en que NO encuentra ninguna solución debido a los valores ocupados en el vector presence, así que devuelve un vector de contenido false en todas sus posiciones. El segundo caso debe devolver una combinación específica, ya que forzamos un valor true en el vector de entrada que, junto con el número de casillas, tiene solo una posible partición única.

Caso 8: interesa testear que llene bien las casillas blancas. Es decir, únicamente con particiones únicas. Puede devolver un tablero con casillas blancas con valor '0'. Significa que una de las dos (o ambas) restricciones de esa casilla no ha sido completada del todo porqué no ha encontrado ninguna combinación. Será entonces cuando omplir_smart deberá actuar y acabar de llenar el tablero.

Caso 9: interesa testear, sobre todo, el control de errores. Por eso, se adjunta un caso en el que la función deberá devolver código de error -1. La parte de llenar la board es completamente aleatoria, pero el error debe ser determinista. El caso es un kakuro inválido porque en la columna número 7 tiene diez casillas blancas seguidas sin ninguna restricción o casilla negra que las separe.

Caso 10: interesa testear y ver que la función detecta los casos de sumas paralelas bien, y no detecta los casos que no tienen. Se adjuntan varios casos. El primero, debe devolver false: tiene dos restricciones de fila seguidas con el mismo valor, pero tienen diferente número de casillas blancas. En el segundo caso también deberá devolver false: tiene dos restricciones de fila seguidas con el mismo número de casillas blancas pero con diferente suma. Los siguientes dos casos deberá devolver true. El primero tiene las dos restricciones mencionadas del caso anterior con la misma suma, así que existe la suma paralela. El segundo, es igual pero ahora tiene la misma casilla de cada restricción con un valor diferente. Deberá poder ignorar los valores escritos en las casillas blancas. El último caso es un tablero que tiene una suma paralela, pero que no tiene las restricciones juntas. Es decir, que existe una fila que las separa (con ninguna casilla negra de por medio). Es un caso más complejo pero que la función debe detectar, y devolver true.

Caso 11: lo mismo que el caso 9; es la misma función. Usaremos este caso para así hacer un output fijo (esperado) de nuestro juego de pruebas.

<u>Formato del input:</u>

Primero, el número de función a evaluar. Enter y esperar los detalles del input que da el Driver. Después de cada dato, Enter. Para finalizar, -1 o número de función a evaluar.

Ejemplo de input en fichero:

```
9
8,8
*,*,C35,C12,*,*,C17,C12
```

Grupo 9.3

,F5,?,?,,C21F4,?,?
*,F10,?,4,F22,?,?,?
,C26F9,?,?,F12,?,?,
F18,?,?,?,F12,?,?,*
F12,5,?,*,*,C3,C10,C7
F7,?,?,*,F13,?,6,?
F12,?,?,*,F7,?,?,?
-1

El número 9 hace referencia al caso. Debe ser un número entre el 0 y el 11.
El String del kakuro debe estar en formato "jutge". Pero puede tener casillas blancas con números entre el 1 y el 9. Depende de qué opción, pueden ser sobreescritos e ignorados.
Al final número de caso para evaluar otra función o -1 para salir del Driver.

- Controlador de Dominio:

La otra parte nueva del testing que hemos hecho para esta tercera entrega es la del Controlador de Dominio. Engloba a los controladores de las otras dos capas. Discriminamos 14 casos diferentes: del número 0 al número 13. Este Driver siempre escribe su salida en un fichero, localizado en /EXE/testingDominio, y donde cada caso escribe en un fichero que se llama según el número de caso.
A continuación, los describimos brevemente:

0. El caso “cero” comprueba las funciones LoadAll y SaveAll. Testea la correcta comunicación de la capa de Dominio con la capa de Persistencia tratando de cargar y guardar todos los objetos relevantes. El objetivo de este caso es ver que los datos se leen y se escriben bien en los ficheros (en la “base de datos”) del sistema.

1. El primer caso testea la función AddProfile. El objetivo es ver que añadimos un usuario al sistema usando esta función y que, efectivamente, se añade correctamente. La manera en la que se ha añadido es mediante su registro.

2. El segundo caso testea dos funcionalidades: añadir Kakuros y añadir Stats (a un Kakuro). Como en el caso anterior, el objetivo es el correcto funcionamiento. Tratamos de añadir un kakuro nuevo en el repositorio de la aplicación. En el caso en el que el kakuro no se encuentre en el repositorio, también deberemos crear las estadísticas del mismo.

3. El tercer caso consiste en añadir un kakuro mediante un fichero. Partimos de un fichero, test.txt, donde escribiremos un kakuro. Queremos ver el correcto uso sobre la carga de un kakuro mediante un fichero. Es indispensable, lógicamente, el path donde se encuentra el fichero.

4. El cuarto caso consiste en generar un kakuro. El testing no se centra en la generación del propio kakuro (para ese caso, ya hay un driver específico), si no en añadirlo correctamente al repositorio de kakuros. Para eso, generaremos un kakuro a partir de un tamaño, de un porcentaje de casillas blancas y de un número de casillas blancas resueltas. Finalmente, lo añadiremos al repositorio y lo escribiremos en el fichero pertinente de este cuarto caso.

5. El quinto caso consiste en eliminar un kakuro. Eliminaremos el primer kakuro que se encuentre en el repositorio. Un punto importante de este caso es ver que con ello se eliminarán también todas las partidas a medias que se encuentren guardadas, para así mantener la consistencia en toda la aplicación.

6. El sexto caso consiste en crear una Partida (Game). Imitamos el comportamiento de un usuario, es decir, seleccionamos un kakuro, hacemos un login, y creamos una partida.

7. El séptimo caso es el de obtener una pista durante una partida. Para el testing, decidimos elegir un kakuro y simular la petición de pistas. Después, se imprime la pista y, finalmente, la solución. De esta manera, se puede ver que la pista realmente está bien dada.

8. El octavo caso consiste en actualizar las estadísticas de un kakuro. Primero, hacemos el login con un usuario que hemos elegido. Después, elegimos un kakuro y lo completamos con un tiempo mínimo, para asegurarnos estar entre los tres mejores tiempos. Finalmente, comprobamos las estadísticas del Kakuro. Estas deberían haber cambiado e insertado el nuevo tiempo correctamente.

9. El noveno caso testeamos la funcionalidad de eliminar una partida. Primero, elegimos un kakuro para jugar. Después, la eliminamos. Es decir, es el propio usuario quien elimina la partida.

10. El décimo caso consiste en cambiar una contraseña. Primero de todo, hacemos login con un usuario que acabamos de crear para el caso. Testeamos que, efectivamente, el password se cambia si se introduce correctamente el password actual. También vemos que, si se introduce erróneamente el password, el programa devuelve error y NO lo modifica.

11. El undécimo caso consiste en obtener la dificultad de un kakuro dada su string. Seleccionamos tres casos de kakuros. El primero es de dificultad media debido a su proporción de casillas blancas. El segundo es difícil debido, también debido a su elevada proporción de casillas blancas. El tercero es fácil debido a su cantidad de casillas resueltas: por su proporción de casillas blancas debería ser de categoría media.

12. El decimosegundo caso consiste en que un usuario elimine su propio perfil. Para lograrlo, primero creamos un perfil, hacemos el login y finalmente lo borramos. Imprimimos el estado de los perfiles por pantalla para comprobar su correcto funcionamiento.

13. El decimotercer caso es más extenso que los anteriores: consiste en comprobar si el kakuro introducido por un usuario, sigue el formato pedido por el programa. La función es isCorrect. A continuación, se explica más a fondo.

- isCorrect:

El testing de esta funcionalidad es mucho más exhaustivo que el de las otras del controlador de dominio porque es más sensible: se encarga de determinar si un kakuro insertado por un usuario es correcto, en cuanto a sintaxis. También hace algunas comprobaciones, como por ejemplo que una fila o columna tengan una suma entre 1 y 45, que haya casillas blancas que no pertenezcan a ninguna restricción de fila o columna, o que haya alguna restricción de fila o columna sin ninguna casilla blanca.

Los juegos de prueba son muy exhaustivos. Se adjunta un fichero, joc_proves_isCorrect.txt. No está pensado para copiarlo y pegarlo directamente en el Driver de ControladorDominio, ya que, por ejemplo, ese driver no acepta más de un 'posible' kakuro por ejecución. El fichero tiene una entrada que pone CORRECTO:, y en la que adjunta diferentes casos de kakuros diferentes (desde un 2x2, trivial, hasta algunos de más grandes), y en los que el programa deberá devolver true. Como caso especial, hemos puesto algún kakuro con casillas blancas resueltas, en los que deberá devolver true. Más abajo en el fichero de texto, hay una segunda entrada que pone NO CORRECTO:, y donde están muchos kakuros que tienen algún defecto y que, por lo tanto, la función deberá devolver false. No se comentarán todos ellos: hay unos 25 más o menos. Lo que sí que se comentará son los casos

explotados con los juegos de prueba. Hemos adjuntado kakuros con menos filas de las que su tamaño indica, con menos columnas de las que debería, con más puntos y a parte (enter) de los que debería, con menos casillas en alguna de sus filas o columnas, con restricciones erróneas (por ejemplo, F0), con sus restricciones de fila i columna giradas (por ejemplo, F18C15 en vez de C15F18), con casillas blancas rodeadas de casillas negras que no son de restricción (es decir, casillas de '*'), casillas blancas que no pertenezcan a ninguna restricción de fila o columna, restricciones de fila o columna sin ninguna casilla blanca, etc.
Procuramos ser bastante exhaustivos con este testing para intentar evitar que un usuario pueda acabar insertando en el sistema un kakuro mal formateado.
Para ejecutar los juegos de prueba, hace falta ejecutar el ControladorDominioDriver, y seleccionar el caso número 13.
NOTA: isCorrect no es la función encargada de detectar si un kakuro tiene una solución o más de una. De esta parte, ya se encarga la función addKakuro (que está en ControladorDominio y que llama a isCorrect), y que se encarga de insertar el kakuro en el sistema.