Informe sobre la implementación del proyecto

Martin agnello, maximo fernandez

2024

**Predicados y sus estrategias**

**verificar\_fila/4**

· Extrae las pistas y la fila correspondiente de la grilla utilizando nth0/3.

El predicado verificar\_fila/4 se encarga de verificar si cumple con las pistas dadas de una fila específica en la grilla utilizando el predicado verifica\_pista/3, para ello Extrae la fila y sus pistas correspondientes de la grilla utilizando nth0/3. Luego verifica si la fila cumple con las pistas dadas con el predicado verifica\_pista/3

**Estrategia**: Primero recuperamos la pista de la fila que vamos a verificar, luego recuperamos la fila en particular y luego llamamos al predicado verificar\_fila\_aux que verifica si cumple con las pistas dadas. sí una fila cumple con sus pistas correspondientes, en caso de que se satisfagan entonces FilaSatisfecha tomara el valor 1, caso contrario tomara el valor 0.

**obtener\_columna/3**

**Estrategia**: primero llamamos a una función que retornara en una lista todos los elementos que se encuentran en la columna deseada de la grilla, luego llamamos a la función invertir para que retorne la columna invertida, esto es para que luego en la función verificar columna se puedan verificar el correcto orden de las pistas con la columna.

**obtener\_columna\_acum/4**

**Estrategia**: Recorremos la lista hasta encontrar el elemento de la columna deseada y luego lo guardamos en la cabeza de la lista a retornar, esto se repite para todas las filas.

**invertir\_lista/2**

**Estrategia**: Llamamos recursivamente con la lista que deseamos invertir y luego en el back tracking concatenamos el primer elemento con la lista invertida.

**verificar\_columna/4**

**Estrategia**: Primero obtenemos la lista de pistas de la columna, luego obtenemos la columna a verificar, una vez obtenidos ambos llamamos a la función que verifica si se cumplen las pistas de la columna con las pistas y la columna recién obtenidas. sí una columna satisface las pistas asignadas a la misma, en caso afirmativo ColumnaSatisfecha tomara el valor 1, caso contrario tomara el valor 0.

**comprobar\_grilla/5**

**Estrategia**: Primero contamos la cantidad de filas y columnas que hay en la grilla, para luego verificar si se satisfacen las pistas de cada una de las filas y de las columnas, luego en caso de que se satisfagan las pistas de todas las filas FilasSatisfechas = 1, caso contrario será igual a 0, es análogo para ColumnasSatistechas.

**comprobar\_todas\_filas/4**

**Estrategia**: Comprueba que todas las filas cumplan las pistas asociadas a ellas, el predicado recibe la cantidad de filas para hacer esa cantidad de iteraciones verificando cada fila según su índice y llamando recursivamente hasta que la cantidad de filas a verificar sean 0. Si se cumple que todas las filas satisfagan sus pistas correspondientes, entonces en caso afirmativo FilaSat = 1, en caso contrario

FilaSat = 0.

**comprobar\_todas\_columnas/4**

**Estrategia**: La estrategia es análoga al predicado comprobar\_todas\_filas/4

**contar\_filas/2**

**Estrategia**: Para hallar la cantidad de filas contamos la cantidad de listas de la grilla.

**contar\_columnas/2**

**Estrategia**: Para hallar la cantidad de columnas contamos la cantidad de elementos de una lista (asumiendo que la cantidad de columnas será igual para todas las filas).

**comprobar\_grilla\_react/5**

**Estrategia**: Este predicado lo utilizamos para obtener las listas de las pistas que se cumplen de las filas y las columnas, para ello primero contamos la cantidad de filas y la cantidad de columnas y luego iteramos esa cantidad de veces respectivamente en cada lista para verificar que pistas se cumplen en cada caso.

**comprobar\_todas\_columnas\_react/4**

**Estrategia**: Se itera una cantColumnas de veces y se verifica que cada columna cumpla sus respectivas pistas, retornara una lista con las pistas que se cumplen de cada columna.

El predicado comprobar\_todas\_columnas\_react/4 se utiliza para retornar la lista con las pistas que se cumplen en cada columna de la grilla.

**comprobar\_todas\_filas\_react/4**

**Estrategia**: Se comporta de manera análoga a comprobar\_todas\_columnas\_react/4 pero con filas.

**verifica\_pista/3**

Dada una lista y sus pistas este predicado retorna un 1 si se cumple la pista y un 0 en caso contrario.

**Estrategia**: Se recorre la lista para ver si cumple la pista, cuando la casilla tiene un elemento distinto de “#” sigue iterando en busca de “#” y cuando encuentra “#” se encarga de verificar que haya la cantidad solicitada por la pista, este proceso se repite para todas las pistas hasta recorrerse toda la lista o hasta cuando no se verifique una pista dada.

**no\_esta\_instanciado/2**

**Estrategia:** Chequea que el elemento sea una lista vacia o una “X” o no este instanciado.

**obtener\_pista/3**

Dado un índice y una lista de pistas, obtiene la pista que se encuentre en el índice.

**verifica\_pistas\_columna/5**

Verifica que se cumplan las pistas de cada columna de la grilla.

**Estrategia**: Se van obteniendo las columnas de la grilla y se van verificando que cumplan con sus respectivas pistas, cuando cumplen se agrega un “1” en una lista de salida indicando que cumple con sus pistas, en caso contrario se agrega un “0” en la lista de salida indicando que no cumple con sus pistas, luego se aumenta en uno el índice y se repite el proceso hasta acabar con las columnas de la grilla.

**todas\_iguales/2**

**Estrategia:** Chequea si el encabezado de la lista es igual al elemento que le pasan por parámetro, luego se llama recursivamente con la cola de la lista y se repite.

**generar\_posibles\_soluciones/2**

Dada una pista retorna una lista que cumpla con la pista dada.

**Estrategia**: Se va rellenando una lista con “#” y se chequea que coincida con la pista, en caso contrario se retrotrae y coloca “X” hasta donde había llegado anteriormente, luego desde donde se quedó vuelve a repetir intentar colocar “#” chequeando que cumpla con las pistas dadas, esto se repite hasta generar la lista deseada.

**fila\_correcta/4**

Dada una lista, busca las combinaciones posibles de las pistas y las intersecta para ver cuales coinciden.

**Estrategia**: Dada una lista, se buscan sus posibles combinaciones (teniendo en cuenta su tamaño), luego se intersectan las listas obtenidas en busca de casillas que siempre coincidan y quedarnos con la lista obtenida por las intersecciones.

**intersección/3**

**Estrategia**: Recibe las posibles combinaciones de una lista y verifica si en todas las listas los elementos son iguales a "X" o "#”. Si son iguales, añade el carácter correspondiente a la lista acumulada. Si no son iguales, añade un “\_”. Esto da como resultado una lista con la intersección de todas las posibles soluciones.

**cumple\_condicion/2**

**Estrategia:** Chequea si la longitud de las pistas más los espacios que habría entre ellas es igual a la longitud L.

**primer\_pasada/4**

**Estrategia**: El predicado recibe la grilla inicial, las pistas de filas y las de columnas, luego para cada fila se ve que pistas se pueden completar y se guarda la fila con las pistas marcadas en una grilla de salida, esto se repite para todas las filas, luego se traspone la grilla y se repite el proceso, pero con las columnas, al final se traspone la grilla de salida una vez más así queda en su posición original.

**segunda\_pasada/4**

**Estrategia:** Si la grilla de entrada ya está completa o sea cumple con todas las pistas de las filas, entonces simplemente devuelve la misma grilla como grilla de salida, caso contrario se genera una grilla con pistas validas, luego se compara la grilla original con la recién generada, si son iguales entonces ya se obtuvo la grilla de salida, sino se vuelve a llamar recursivamente.

**grillas\_iguales/2**

**Estrategia**: Determina si 2 grillas son iguales comparando las filas de cada una y llamando recursivamente con las colas de ambas listas.

**filas\_iguales/2**

**Estrategia**: Recorre 2 listas y va chequeando si el elemento que encabeza ambas listas es el mismo, tanto si es “#” , “X” o “\_” en ambas.

**grilla\_completa/2**

Verifica si todas las filas de una grilla están completamente instanciadas

**Estrategia**: Se obtiene la última fila de la grilla y se chequea que esté completamente instanciada, luego llama recursivamente con el resto de la grilla, disminuyendo el índice hasta que llegue a 0 que será cuando no haya más filas por recorrer.

**elementos\_instanciados/1**

**Estrategia**: Verifica que todos los elementos de una lista esten instanciados.

**grilla\_correcta/4**

**Estrategia:** Dada la grilla, las pistas de filas y las de columnas, recorre todas las filas y guardan las pistas que se puedan afirmar en una grilla de salida, luego se traspone la grilla obtenida y se repite el proceso, por ultimo se traspone nuevamente la grilla.

**generar\_filas\_correctas/5**

**Estrategia**: Se va recorriendo la grilla, luego para cada fila y su pista correspondiente se genera una fila de salida con las pistas que se pueden confirmar como validas, luego se repite el proceso para todas las filas hasta llegar al total de filas de la grilla, esto se controla con un índice que se va aumentando en cada iteración.

**ultima\_pasada/4**

**Estrategia:** En el primer caso recursivo va recorriendo las filas de a una, si todos los elementos de la fila no son variables, entonces agrega esta fila a la lista acumulada, y llama recursivamente a con las filas restantes, las pistas restantes, y la lista acumulada actualizada.

En el segundo caso recursivo genera las posibles soluciones para la fila actual usando las pistas correspondientes, entonces agrega esta fila a la lista acumulada, y llama recursivamente a con las filas restantes, las pistas restantes para las filas, y la lista acumulada actualizada.

Por último, si no quedan más filas por procesar, verifica que las columnas de la grilla acumulada cumplan con las pistas de las columnas. Si todas las columnas son válidas, la grilla resultante será la grilla acumulada.

**solución/4**

**Estrategia:** Dada la grilla inicial, las pistas de filas y columnas, genera una solución valida utilizando la grilla ya procesada de primera\_pasada/4 como dato de entrada para segunda\_pasada/4 este a su vez, nos dará la grilla aun con más pistas marcadas, luego esta grilla se utiliza en ultima\_pasada/4 que devuelve grilla verificada con todas las pistas verificadas.

**Funcionalidades extra implementadas**

Se agregó la posibilidad de reiniciar el nivel, en caso de que el usuario desee reiniciarlo por cualquier razón, va a poder hacerlo gracias al botón “Reiniciar juego” que reestablece el estado del juego a su estado inicial.

**Aspectos positivos de la resolución**

Los aspectos positivos del proyecto fueron el aprendizaje de React, así también como el aprendizaje de otros lenguajes como JavaScript y html en rasgos generales.

Otro aspecto positivo fue la eficiencia de la gran mayoría de predicados que escribimos en Prolog.

**Desafíos encontrados**

Uno de los principales desafíos fue el de no saber cómo empezar el proyecto, había mucho por hacer, varias carpetas, varios archivos de distintas extensiones, mucho código y al ser la primera vez que veíamos algo así se sintió bastante abrumador, una vez decididos sobre donde arrancar, que fue por el archivo prolcc.pl también fue difícil el pensar cómo se iba a conectar todo el programa, esto último nos tomó un buen tiempo debido a que no sabíamos cómo continuar, cosa que con el tiempo y consultas fuimos arreglando.

El conseguir una columna en particular de la grilla fue un desafío bastante interesante y la solución que propusimos nos pareció dentro de lo que habíamos visto bastante llamativa por cómo funciona.

Programar en React también ha sido un desafío, como por ejemplo el cómo conectar si una pista está bien o mal con su color.

**Correcciones de la primera entrega**

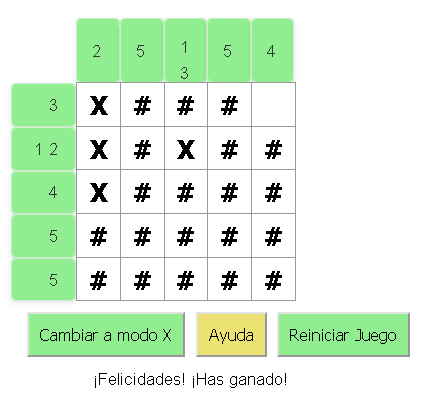
Se eliminaron las apariciones de variables singleton. Ahora no hay que presionar un botón para saber si se ganó el juego. Y ahora no se puede seguir interactuando con el Nonograma una vez ganado.

**Correcciones de la segunda entrega**

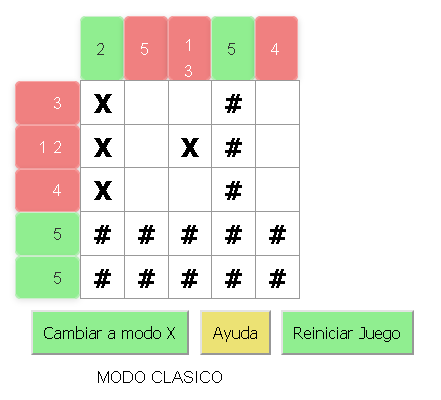
Ahora la solución funciona para múltiples configuraciones iniciales, incluida la que se dio como ejemplo en la corrección y no crashea al intentar revelar una celda, se eliminaron cuts innecesarios en el código, se colocaron los valores de “#” y “X” en el predicado de generar\_posibles\_soluciones/2, ahora se da una explicación mas completa de los predicados mas importantes.

**Casos de Test**

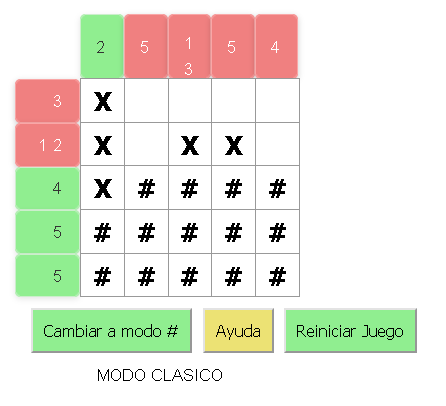
En caso de ganar el juego y presionar el botón “Comprobar” se chequea que haya ganado el juego y lo informa.



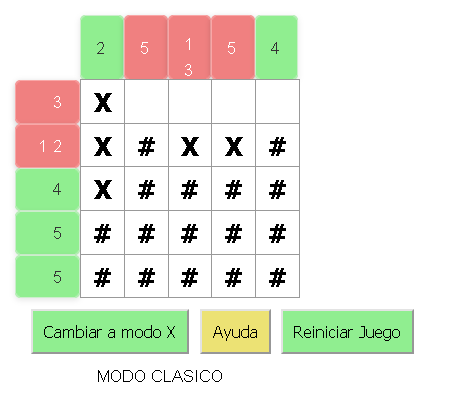
A medida que se cumplen las pistas de una fila o columna las mismas se van marcando en verde, caso contrario en rojo.



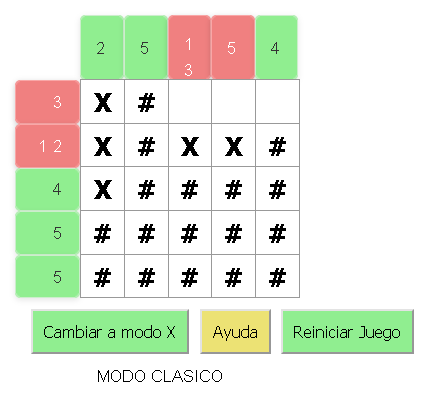
En caso de presionar el botón “Cambiar a modo X” el juego pasa al modo colocar “X” y también se modifica el botón a “Cambiar a modo #”.



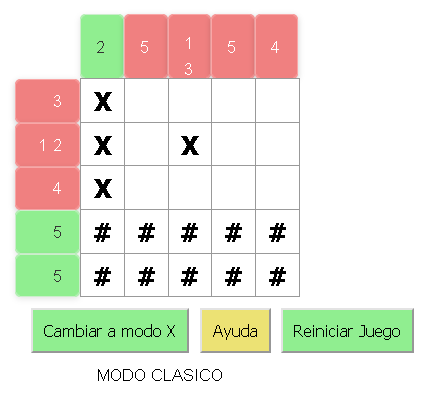
En caso de presionar el botón “Cambiar a modo #” el juego pasa al modo colocar “#” y también se modifica el botón a “Cambiar a modo X”.



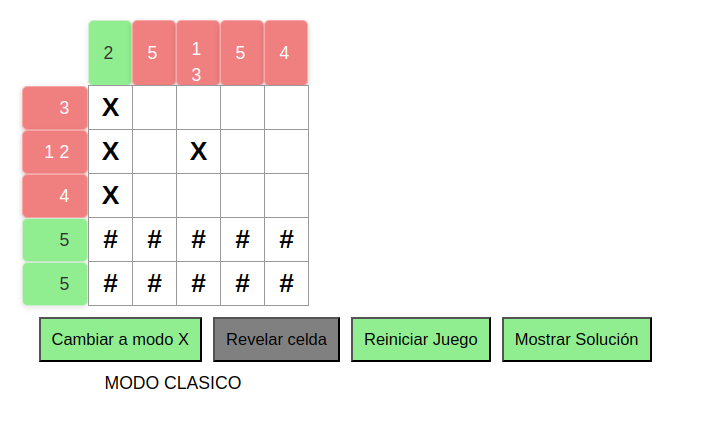
En caso de presionar el botón “Reiniciar juego” el juego volverá a su estado inicial. Estado del juego antes de presionar el botón “Reiniciar juego”:



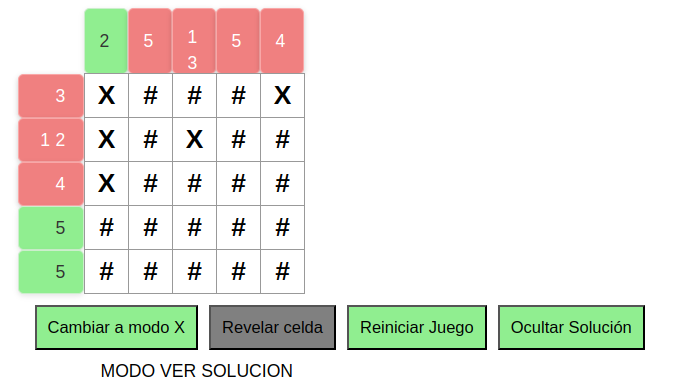
Estado del juego luego de presionar el botón “Reiniciar juego”:



En caso de presionar el botón “Mostrar Solucionar” el juego revelara la solución del Nonograma, también se modifica el botón a “Ocultar Solución”.

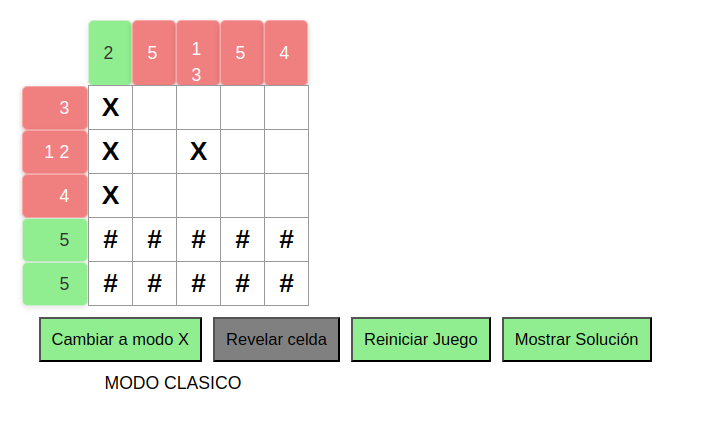
Antes de presionar el botón:

Luego de presionar el botón:



En caso de presionar el botón “Revelar celda” en una celda vacía, se mostrará el contenido de la celda acorde a la solución, y el botón se modificará de color gris a color verde.

Antes de presionar el botón:



Luego de presionar el botón:

