2024

Informe sobre la implementación del proyecto

MARTIN AGNELLO, MAXIMO FERNANDEZ

Predicados y sus estrategias

verificar_fila/4

El predicado verificar_fila/4 se encarga de verificar si se cumplen las pistas dadas de una fila específica en la grilla utilizando el predicado verifica_pista/3, para ello se extrae la fila y sus pistas correspondientes de la grilla utilizando nth0/3. Luego se verifica si la fila cumple con sus pistas con el predicado verifica_pista/3 explicado a continuación, sí una fila satisface las pistas asignadas a la misma, en caso afirmativo Cumple tomara el valor 1, caso contrario tomara el valor 0.

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.

obtener_columna/3

Estrategia: primero llamamos a obtener_columna_acum/4 que retornara en una lista todos los elementos que se encuentran en la columna deseada de la grilla, luego llamamos a invertir_lista/2 para que retorne la columna invertida, esto es para que luego en verificar_columna/4 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 con append/3 el primer elemento con la lista invertida.

verificar_columna/4

Estrategia: Primero obtenemos la lista de pistas de la columna con ntho/3, luego obtenemos la columna a verificar con obtener_columna/3, una vez obtenidos ambos llamamos a verifica_pista/3 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 Cumple tomara el valor 1, caso contrario tomara el valor 0.

comprobar_grilla/5

Estrategia: Primero contamos la cantidad de filas y columnas con contar_filas/2 y contar_columnas/2 que hay en la grilla, para luego verificar si se satisfacen las pistas de cada una de las filas y de las columnas con comprobar_todas_filas/4 y comprobar_todas_columnas/4, luego en caso de que se satisfagan las pistas de todas las filas FilaSat = 1, caso contrario será igual a 0, es análogo para ColSat.

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 con contar_filas/2 y la cantidad de columnas con contar_columnas/2 y luego iteramos esa cantidad de veces respectivamente en cada lista para verificar que pistas se cumplen en cada caso, obteniendo así dos listas con las pistas que se cumplen, una con las pistas de las filas y otra con las pistas de las columnas que contendrán 1s y 0s, cada 1 significa que una lista cumplió con sus pistas y cada 0 significa que una lista no cumplió con sus pistas .

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.

no_esta_instanciado/1

Estrategia: Chequea que el elemento sea una lista vacía o una “X”, o no este instanciado utilizando el predicado ground/1 para ver si el elemento está instanciado y luego negando el resultado.

obtener_pista/3

Dado un índice y una lista de pistas, obtiene la pista que se encuentre en el índice utilizando nth0/3.

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 con el predicado obtener_columna/3 y se van verificando que cumplan con sus respectivas pistas con el predicado verifica_pista/3, 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 que esto ocurre al llega al caso base que es cuando el índice llega a la misma cantidad de columnas de la grilla y nos quedamos sin pistas.

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 interseca para ver cuales coinciden.

Estrategia: Dada una lista, se buscan sus posibles combinaciones (teniendo en cuenta su tamaño) con generar_posibles_soluciones/2, luego se intersecan con interseccion/3 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 con interseccion_aux/4 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 con primera_pasada_aux/4 se ve que pistas se pueden completar con cumple_condicion/2 y fila_correcta/4 y se guarda la fila con las pistas marcadas que se pueden afirmar en una grilla de salida, esto se repite para todas las filas, al finalizar el recorrido de todas las filas, se traspone la grilla con transpose/2 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 (esto se se verifica con grilla_completa/1) , entonces simplemente devuelve la misma grilla como grilla de salida, caso contrario se genera una grilla con pistas validas utilizando grilla_correcta/4, luego se compara la grilla original con la recién generada utilizando grillas_iguales/2, si son iguales entonces ya se obtuvo la grilla de salida, sino se vuelve a llamar recursivamente.

grilla_completa/1

Verifica si todas las filas de una grilla están completamente instanciadas

Estrategia: Se va recorriendo la grilla comenzando desde la cabeza de la grilla y se chequea que la fila actual este completamente instanciada utilizando el predicado elementos_instanciados/1, luego llama recursivamente con el resto de la grilla hasta que no haya más filas por recorrer.

grilla_correcta/4

Estrategia: Dada la grilla, las pistas de filas y las de columnas, recorre todas las filas y se guardan las pistas que se puedan afirmar en una grilla de salida utilizando generar_filas_correctas/3, luego se traspone la grilla obtenida con transpose/2 y se repite el proceso para las columnas, por último, se traspone nuevamente la grilla.

generar_filas_correctas/3

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 utilizando fila_correcta/4, luego se repite el proceso para todas las filas hasta llegar al total de filas de la grilla.

grillas_iguales/2

Estrategia: Determina si 2 grillas son iguales comparando las filas de cada una utilizando filas_iguales/2 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.

ultima_pasada/4

Estrategia: Utiliza ultima_pasada_aux/5 que en el primer caso recursivo va recorriendo las filas de a una, verificando si todos los elementos de la fila no son variables con forall/2, member/2 y nonvar/1, luego agrega esta fila a la lista acumulada con append/3, 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 con generar_posibles_soluciones/3, entonces agrega esta fila a la lista acumulada con append/3, 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 con verifica_pistas_columna/5. 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 grilla con listas de celdas validas, luego utiliza 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 celdas validas, luego esta grilla se utiliza en ultima_pasada/4 que devuelve la grilla con todas las celdas validas, asegurándose de que todas las pistas de las filas y las columnas se cumplan completamente.

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 restablece 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

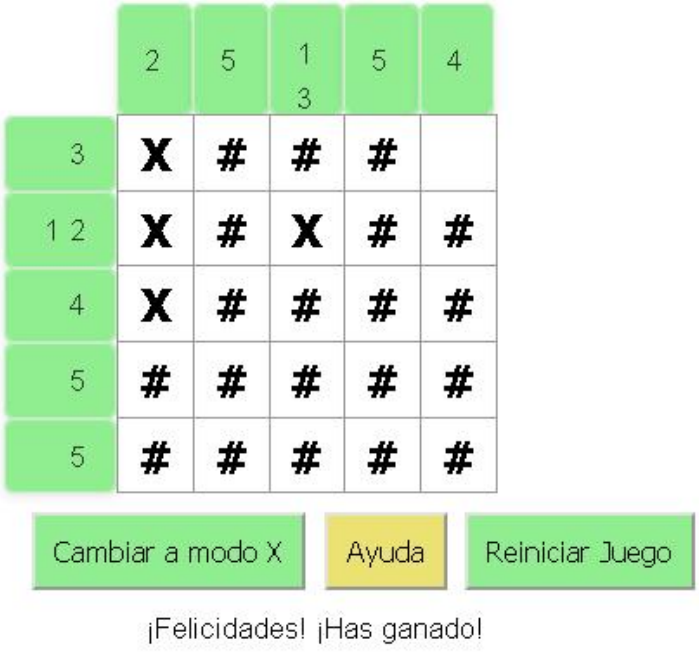
Se eliminaron las apariciones de variables singleton. Ahora no hay que presionar un botón para saber si se ganó el juego. Y ahora no se puede seguir interactuando con el Nonograma una vez ganado.

Correcciones de la segunda entrega

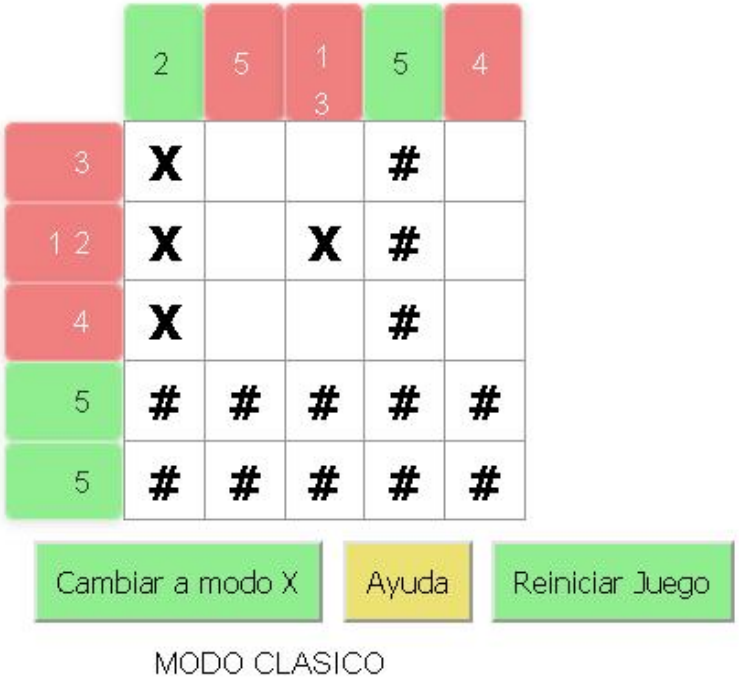
Ahora la solución funciona para múltiples configuraciones iniciales, incluida la que se dio como ejemplo en la corrección y no crashea al intentar revelar una celda, se eliminaron cuts innecesarios en el código, se colocaron los valores de “#” y “X” en el predicado de generar_posibles_soluciones/2 y en otros predicados donde se trabajaba de forma similar, se itera en generar_filas_correctas/5 y en grilla_completa/1 con la cabeza y cola de la lista en lugar de utilizar índices y ahora se da una explicación más completa de los predicados más importantes.

Casos de Test

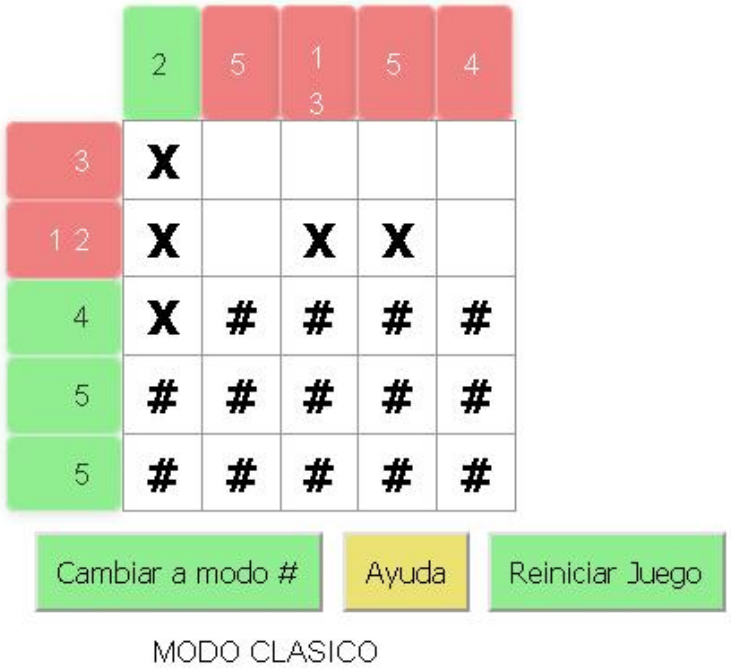
En caso de ganar el juego y presionar el botón “Comprobar” se chequea que haya ganado el juego y lo informa.



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

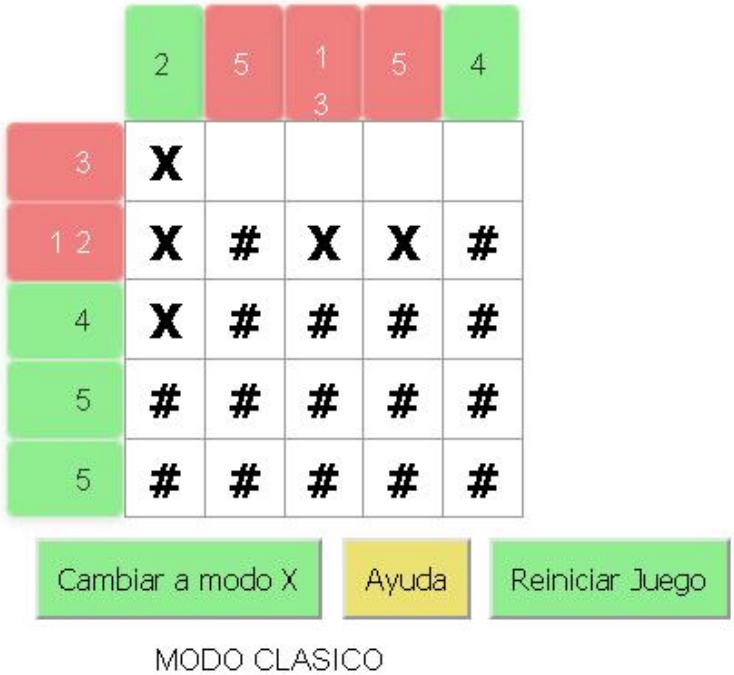


En caso de presionar el botón “Cambiar a modo X” el juego pasa al modo colocar “X” y también se modifica el botón a “Cambiar a modo #”.



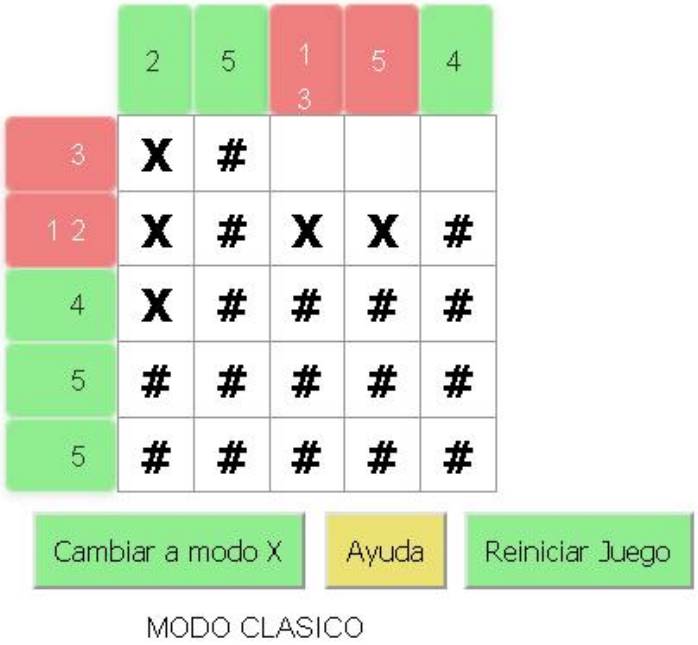
MODO CLASICO

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”.

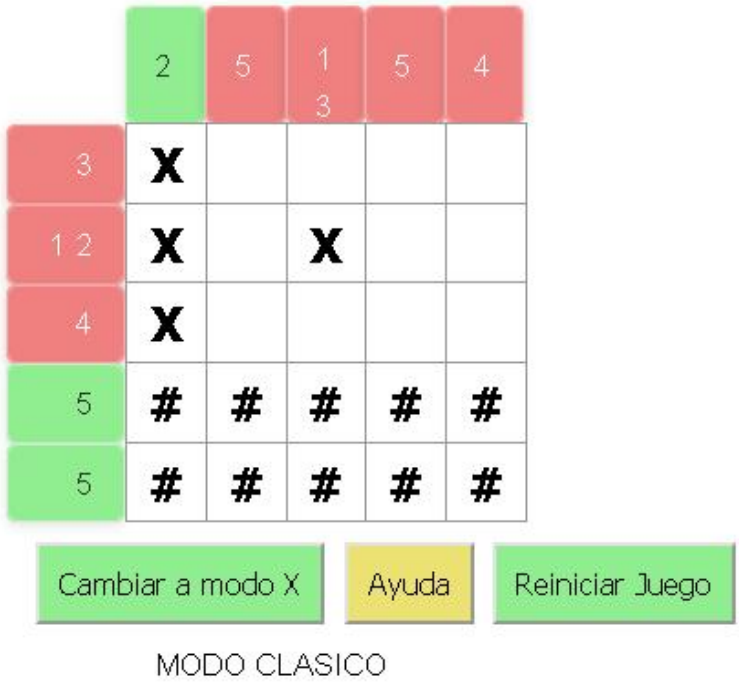


MODO CLASICO

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”:

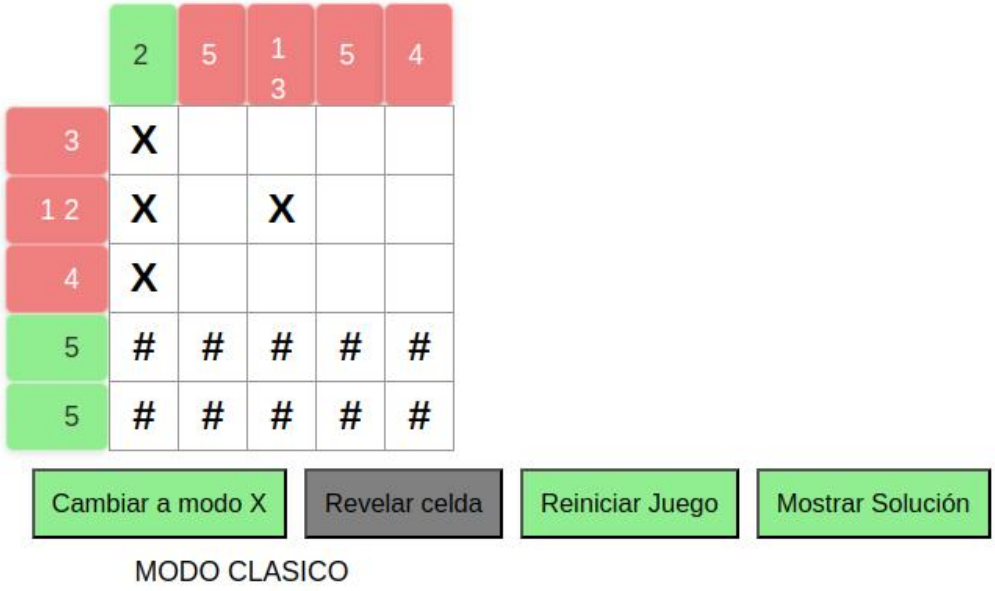


Estado del juego luego de presionar el botón “Reiniciar juego”:



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:



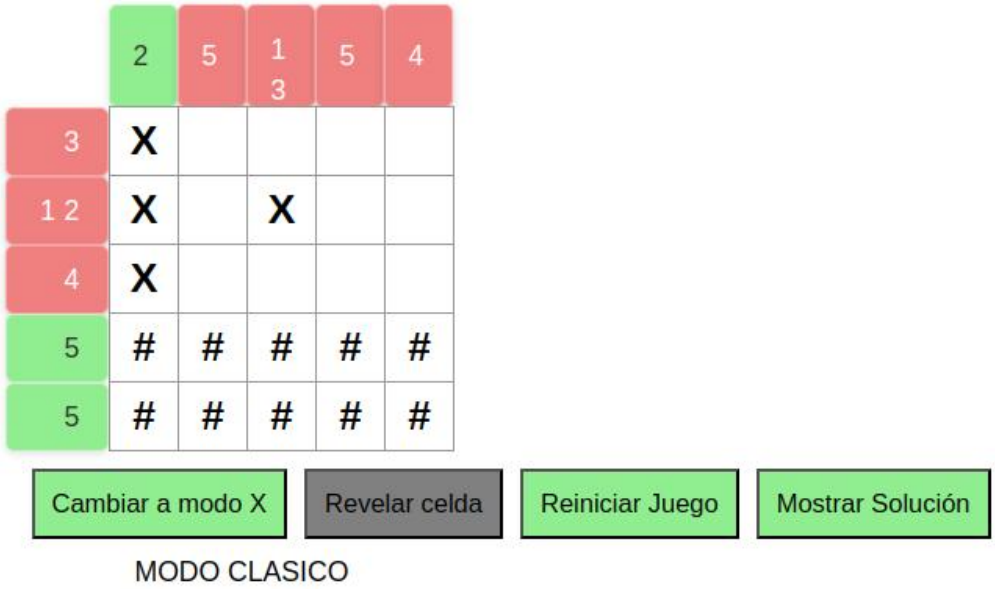
MODO CLASICO

Luego de presionar el botón:

MODO VER SOLUCION

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: