Informe Proyecto

Valentino Villar - Alejo Quintana
Comisión Nº15

Chequeo de pistas

La estrategia de chequeo principal la pensamos de la siguiente manera. Hicimos dos predicados, el primero ***search_clues/3*** para recorrer la lista hasta el primer "#" y el segundo predicado ***check_clue/3*** se encarga de chequear a partir de ese "#" las pistas consecutivas, para luego volver a llamar al primer predicado. De esta manera logramos resolver la estrategia principal del juego, la cual consistió en corroborar si cumplía con las pistas dadas una determinada fila o columna.

Luego dichos predicados se utilizaron dentro de ***put/9*** para completar el propósito del predicado mencionado. El mismo empieza reemplazando en las coordenadas dadas el contenido especificado, luego nos encargamos de buscar las pistas correspondientes a la fila y columna ingresada, para por último comprobar si las pistas de la fila y de la columna fueron satisfechas. Además agregamos el predicado ***count_grid/2*** el cual cuenta la cantidad de "#" en la grilla.

Chequeo inicial y victoria

También se agregó un chequeo inicial ***initial_check/6***, el cual nos permite comprobar si al inicio del juego ya habían pistas satisfechas. El mismo va a ser reutilizado en otro predicado

para comprobar si todas las pistas fueron satisfechas, es decir, se ganó el juego.

En cada inserción en la grilla nos encargamos de chequear con el predicado ***victory_check/3***, si la grilla fue resuelta. Reutilizando el predicado mencionado en el párrafo anterior el cual nos devuelve dos listas que nos indican las pistas válidas en las filas y columnas. Para luego con el predicado ***is_valid/2*** poder comprobar si ambas listas son completamente *true*.

Cambio de modo

Para seleccionar el modo en el que el usuario quiere estar (si quiere colocar una X o pintar la celda) se le brinda un switch que le permite cambiar de un modo a otro. Según el modo en el que se encuentre el Switch se decidirá si se coloca una X o si se pinta la celda. A nivel código, lo que hace el switch es modificar el parámetro content que se le envía al método put, cambiándolo constantemente entre "X" o "#", ya que las celdas pintadas se identifican con un "#". En caso de que en la celda clickeada ya haya un símbolo del mismo tipo este es eliminado, y en caso de ser de otro tipo, es reemplazado. A continuación se muestran los dos posibles estados del Switch y el código asociado que modela el comportamiento deseado:


```
const content = painting ? '#' : 'X'; // Content to put in the clicked square.
<SwitchBtn painting={painting} onClick={() => setPainting(!painting)}/>
```

Resolución total de la grilla

Para la segunda etapa del proyecto se implementó la funcionalidad de resolver la grilla por completo, la cual la pensamos de una forma similar al chequeo. Utilizando dos predicados, en donde el primero es ***combinations/3*** el cual tiene dos caminos, avanzar un lugar en la lista o empezar a poner las pistas, es decir llamar al segundo predicado ***consecutive/3***, que, como bien dice el nombre, se encarga de colocar todas las pistas seguidas. Por lo tanto con estos dos predicados comunicándose entre sí, logramos encontrar la forma de poder crear todas las soluciones dada una lista y sus pistas correspondientes. Ambos predicados tienen en cuenta si ya estaba colocado un "#" o un "X" para crear las combinaciones.

La estrategia presentada anteriormente se contempla únicamente para una lista, por lo que tendremos que adaptarla a una grilla. Para eso se utilizó el predicado ***combinations_grid/3*** el cual, dada la primera fila, llama a ***combinations/3*** y luego recursivamente a sí mismo para que abarque a todas las filas y se vaya armando el árbol de recursión de las distintas posibilidades.

Dicho tablero completo se puede contemplar presionando el siguiente botón:

Revelar en la grilla

Para revelar al usuario una celda de la grilla en particular cuando este lo solicite, se hace uso de la misma grilla que se utiliza a la hora de mostrar la solución total del nonograma. Se busca la celda solicitada por el usuario y se le incorpora a la grilla el símbolo correcto de esa celda. Dicha funcionalidad se obtiene al presionar el siguiente botón:

Casos de prueba

Algunos de las grillas/niveles que se utilizaron para comprobar el correcto funcionamiento del sistema fueron:

Caso de prueba 10x5.

```
init(
[[2,2],[5],[5],[1,1,1],[1,1],[2,2],[5],[5],[1,1,1],[1,1]], % RowsClues
[[10],[3,3],[3,3],[3,3],[10]], % ColsClues
[[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ], % Grid
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],
[ _ , _ , _ , _ , _ ],]).
```

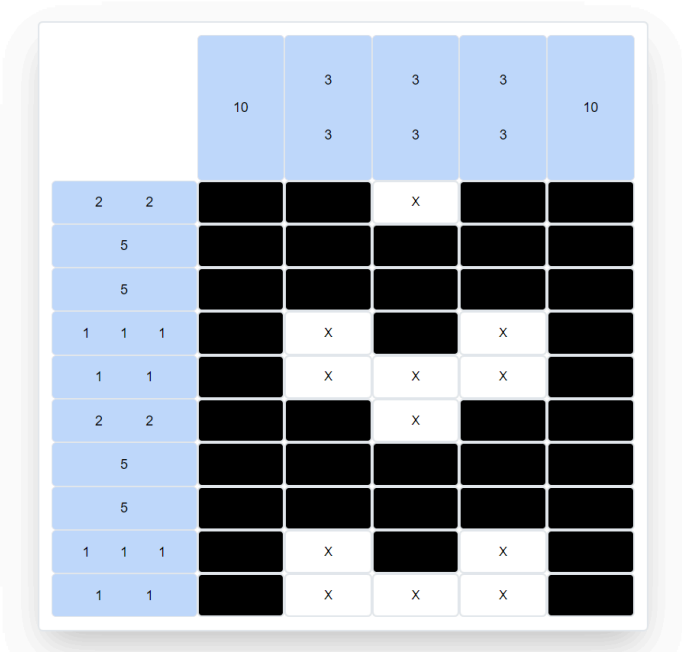
NONOGRAM.

0/38

Reveal

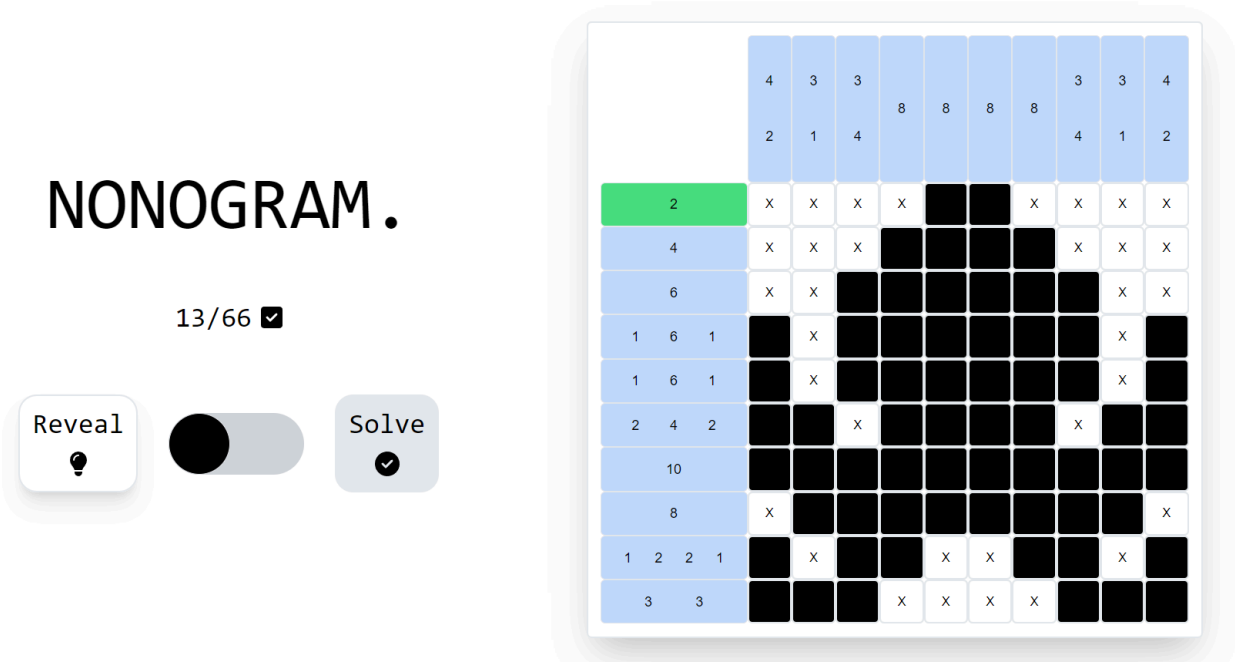
X

Solve



Caso de prueba 10X10 --- Pulpo

```
init([[2], [4], [6], [1,6,1], [1,6,1], [2,4,2], [10], [8], [1,2,2,1],
[3,3]],
     [[4,2], [3,1], [3,4], [8], [8], [8], [8], [3,4], [3,1], [4,2]],
     [[ _ , _ , _ , _ ,"#","#", _ , _ , _ , _ ],
     [ _ , _ , _ ,"#", _ , _ ,"#", _ , _ , _ ],
     [ _ , _ , _ , _ , _ , _ , _ , _ , _ , _ ],
     ["#", _ , _ , _ ,"#","#", _ , _ , _ , _ ],
     ["#", _ , _ , _ , _ , _ , _ , _ , _ , _ ],
     [ _ , _ , _ , _ , _ , _ , _ , _ , _ , _ ],
     [ _ , _ , _ , _ , _ , _ , _ , _ , _ ,"#"],
     [ _ , _ , _ , _ , _ , _ , _ , _ , _ , _ ],
     [ _ , _ ,"#","#", _ , _ , _ , _ , _ ,"#"],
    [ _ , _ , _ , _ , _ , _ , _ , _ , _ ,"#"]
   ]).
```



Extras

Como extra podemos mencionar la utilización del Framework Tailwind CSS que nos permite aplicar estilos mucho más sencillo y rápido.

Durante toda la partida contamos con un contador el cual nos dice cuántos bloques pintados llevamos, y nos da una pequeña pista diciéndonos cuantos bloques pintados tiene la solución.

Al final del juego cuando el jugador gana, se le emite en pantalla una alerta indicando que ganó con un botón que le permite reiniciar el nivel y comenzar de nuevo.

Correcciones Proyecto 1

Entre las correcciones del Proyecto 1 cabe mencionar que se modificó la estrategia utilizada en el *check_clue/3* ya que resultaba un predicado bastante engorroso y difícil de entender.
Pudimos resolver los problemas generados por el backtracking, entendiendo en donde se causaba, cómo evitarlo y como usarlo a nuestro favor.
Utilizamos un predicado para corroborar si una lista era válida por completo, en vez de hacer ambos chequeos en un mismo predicado.
Nos dimos cuenta que el predicado para manejar la victoria era muy similar al chequeo inicial, por lo tanto optamos por hace reutilización del mismo.