**Informe sobre**

**la implementación en Prolog**

Formato del documento primero se mostrará el predicado con sus datos de entrada y datos de salida, luego detallamos la estrategia con la que resolvemos el problema que soluciona el predicado y luego describimos que funcionalidad tiene cada predicado en particular.

**verificar\_fila(+IndiceFila, +PistasFilas, +Fila, -FilaSatistecha)**

**Estrategia:** Primero recuperamos la pista de la fila que vamos a verificar, luego recuperamos la fila en particular y luego llamamos a una función que verifica si cumple con las pistas dadas.

**verificar\_fila(+IndiceFila, +PistasFilas, +GrillaRes, -FilaSat)**

En el predicado verificar\_fila/4 se utiliza para verificar si una fila cumple con sus pistas correspondientes, en caso de que se satisfagan entonces FilaSat tomara el valor 1.

**verificar\_fila(+\_, +\_, +\_, -0)**

El predicado verificar\_fila/4 con el 0 como ultimo parámetro se complementa con el predicado anterior, este predicado es útil cuando no se cumple que la fila satisfaga las pistas dadas, por lo tanto, FilaSat tomara el valor 0.

**obtener\_columna(+Grilla, +NumColumna, -Columna)**

**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(+Grilla, +Col, -Columna)**

El predicado obtener\_columna/3 se utiliza para obtener la columna de una matriz

**obtener\_columna\_acum(+Grilla, +NumColumna, -ColumnaAcum, -Columna)**

**Estrategia:** Recorremos la lista hasta encontrar el elemento de la columna deseada y luego lo guardamos en la cabeza de la lista a retornar

**obtener\_columna\_acum(+[Fila | Grilla], +Col, -ColumnaAcum, -Columna)**

El predicado auxiliar obtener\_columna\_acum/4 se utiliza para ir acumulando los elementos de una determinada columna de una matriz, este predicado es utilizado por obtener\_columna/3.

**obtener\_columna\_acum(+ [ ], +\_, +ColumnaAcum, -ColumnaAcum)**

El predicado auxiliar obtener\_columna\_acum/4 es el caso base del predicado anterior, cuando no hay matriz que recorrer entonces la columna acumulada va a ser la columna a retornar.

**invertir\_lista(+ListaOriginal, -ListaInvertida)**

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

**invertir\_lista(+[X|Xs], -ListaInvertida)**

El predicado invertir\_lista/2 se utiliza para invertir una lista no vacía.

**invertir\_lista(+ [], -[]).**

El predicado invertir\_lista/2 es el caso base del predicado anterior, invertir una lista vacía es igual a una lista vacía.

**verificar\_columna(+IndiceColumna, +PistasCol, +Grilla, -ColumnaSatisfecha)**

**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.

**verificar\_columna(+IndiceColumna, +PistasCol, +GrillaRes, -ColSat)**

El predicado verificar\_columna /4 se utiliza para verificar si una columna satisface las pistas asignadas a la misma, en caso afirmativo ColSat tomara el valor 1

**verificar\_columna(+\_,+\_,+\_,-0).**

El predicado verificar\_columna /4 con el 0 como ultimo parámetro se complementa con el predicado anterior, este predicado es útil cuando no se cumple que la columna satisfaga las pistas dadas, por lo tanto, ColSat tomara el valor 0.

**verificar\_pistas\_en\_lista(+ListaPistas, -ListaFilas)**

**Estrategia:** Si es que hay pistas, se verifica que la primera casilla esta pintada, si lo está, llama a la función verificar\_pconsecutivos para verificar si la pista en cuestión esta pintada y luego llama recursivamente con el resto de la lista no recorrida, en caso de que la primera casilla no está pintada llama recursivamente con el resto de la lista, y en caso de que no haya más pistas que chequear, pero si hay lista por recorrer, entonces verifica que no haya ninguna casilla pintada.

**verificar\_pistas\_en\_lista(+[X|Pistas], -[Y|ListaFilaS])**

El predicado verificar\_pistas\_en\_lista/2 se utiliza para verificar que se cumplan las pistas dadas de una determinada lista

**verificar\_pistas\_en\_lista(+Pistas, -[Y|ListaFilaS])**

El predicado verificar\_pistas\_en\_lista/2 se utiliza para verificar que se cumplan las pistas dadas de una determinada lista

**verificar\_pistas\_en\_lista(+[], -ListaFila)**

El predicado verificar\_pistas\_en\_lista/2 es el caso base de los predicados anteriores, si no hay más pistas que verificar entonces verifica que no haya más casillas pintadas de las que debería.

**verificar\_pconsecutivos(+N, +Lista, -ListaRestante)**

**Estrategia:** Verifica que haya exactamente n casillas consecutivas pintadas.

**verificar\_pconsecutivos(+N, +[X|Filarestante], -Filarestante2)**

El predicado verificar\_pconsecutivos/3 verifica si hay exactamente n casillas consecutivas pintadas, cuyo caso afirmativo retorna la porción de lista que falta recorrer.

**verificar\_pconsecutivos(+0, +[], -[]).**

El predicado verificar\_pconsecutivos/3 es el caso base 1 del predicado anterior, si no hay mas pistas, ni lista entonces se cumple que haya n casillas consecutivas pintadas y se retorna la lista vacía.

**verificar\_pconsecutivos(+0, +[X|Filarestante], -Filarestante)**

El predicado verificar\_pconsecutivos/3 es el caso base 2 del predicado verificar\_pconsecutivos(+N, +[X|Filarestante], -Filarestante2), si no hay más pistas, y la lista es no vacía entonces seguido de las casillas pintadas no debe haber otra casilla pintada.

**comprobar\_grilla(+Grilla, +PistasFilas, +PistasCol, -FilasSatistechas, -ColumnasSatistechas)**

**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\_grilla(+Grilla, +PistasFilas, +PistasCol, -FilaSat, -ColSat)**

El predicado comprobar\_grilla/5 se utiliza para verificar si se gano el juego, verifica que cada fila y cada columna cumpla con sus respectivas pistas, en caso afirmativo para las filas retornara FilaSat = 1, caso contrario FilaSat = 0, es análogo para ColSat.

**comprobar\_todas\_filas(+Grilla, -FilasSatisfechas, +PistasFilas, +CantFilas)**

**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.

**comprobar\_todas\_filas(+Grilla, -FilaSat, +PistasFilas, +CantFilas)**

El predicado comprobar\_todas\_filas/4 se utiliza para verificar si se cumple que todas las filas satisfagan sus pistas correspondientes, en caso afirmativo FilaSat = 1, en caso contrario FilaSat = 0.

**comprobar\_todas\_filas(+\_, +\_, +\_, -0)**

El predicado comprobar\_todas\_filas/4 se utiliza como caso sabe del predicado anterior, si no hay mas filas que recorrer entonces se cumplen las pistas dadas.

**comprobar\_todas\_columnas(+Grilla, -ColSat, +PistasCol, +CantColumnas)**

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

**comprobar\_todas\_columnas(+Grilla, -ColSat, +PistasCol, +CantColumnas)**

El predicado comprobar\_todas\_columnas /4 se utiliza para verificar si se cumple que todas las columnas satisfagan sus pistas correspondientes, en caso afirmativo

ColSat = 1, en caso contrario ColSat = 0.

**comprobar\_todas\_columnas(+\_, -\_, +\_, +0)**

El predicado comprobar\_todas\_filas/4 se utiliza como caso sabe del predicado anterior, si no hay más filas que recorrer entonces se cumplen las pistas dadas.

**contar\_filas(+Grilla, -Cont)**

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

**contar\_filas(+[H|T], -Cont)**

El predicado contar\_filas/2 se utiliza para contar la cantidad de filas de una grilla.

**contar\_filas(+[], -0).**

El predicado contar\_filas/2 se utiliza como caso base del predicado anterior, retorna la cantidad de elementos de una lista vacía que es 0.

**contar\_columnas(+[CabezaLista |\_], -Cont)**

**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).

**contar\_columnas(+[H|\_], -Cont)**

El predicado contar\_columnas /2 se utiliza para contar la cantidad de columnas de una grilla.

**contar\_columnas(+[],- 0).**

El predicado contar\_columnas /2 se utiliza como caso base del predicado anterior, retorna la cantidad de elementos de una lista vacía que es 0.

**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**

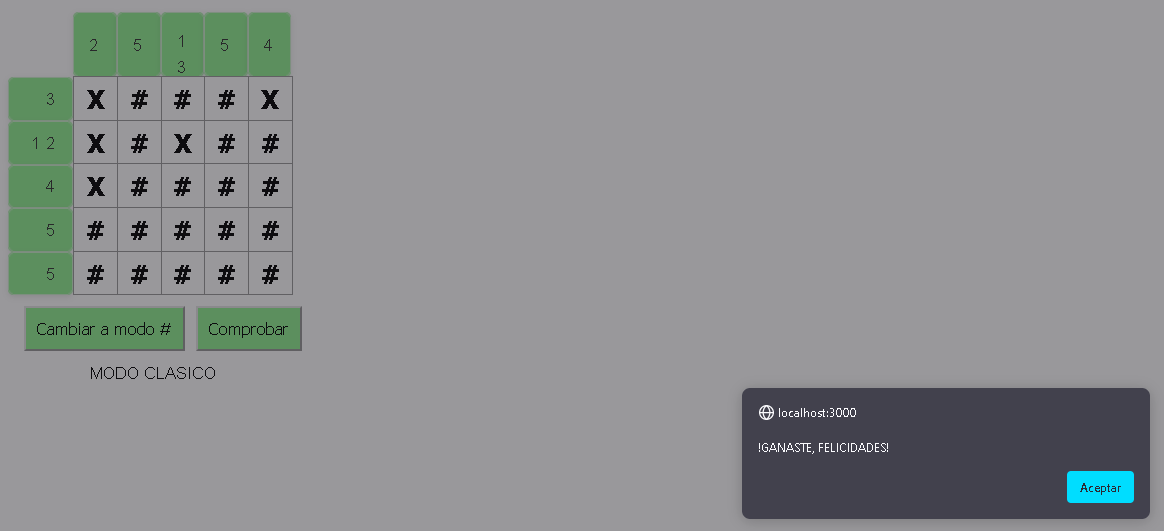
Uno de los principales desafíos fue el de no saber cómo empezar el proyecto, había mucho por hacer, varias carpetas, varios archivos de distintas extensiones, mucho código y al ser la primera vez que veíamos algo así se sintió bastante abrumador, una vez decididos sobre donde arrancar, que fue por el 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.

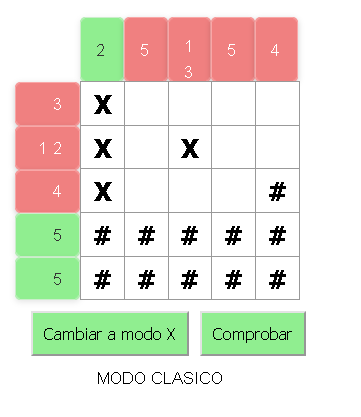
Programar en React también ha sido un desafío, como por ejemplo el cómo conectar si una pista está bien o mal con su color.

**Casos de Test**

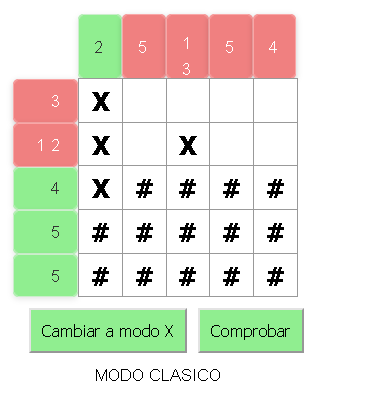
En caso de ganar el juego y presionar el botón “Comprobar” se chequea que haya ganado el juego y lo informa.



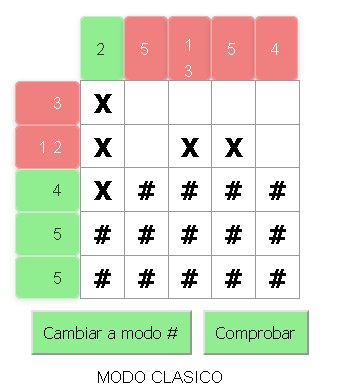
A medida que se cumplen las pistas de una fila o columna las mismas se van marcando en verde, caso contrario en rojo.



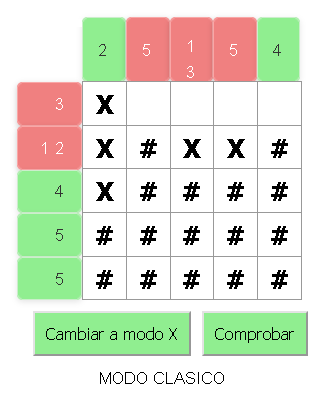
En caso de presionar el botón “Comprobar” y que el juego no este terminado no sale ningún cartel y se podrá seguir jugando.



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 #”.

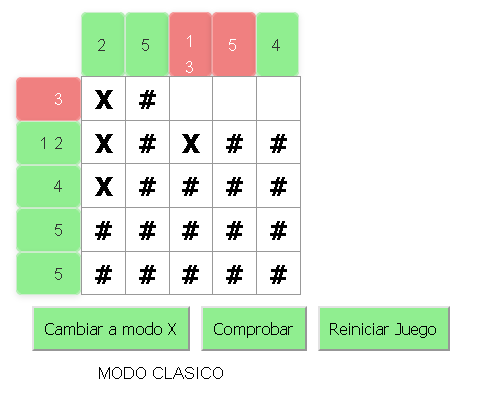


En caso de presionar el botón “Cambiar a modo #” el juego pasa al modo colocar “#” y también se modifica el botón a “Cambiar a modo X”.



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”:

