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. Luego se muestran las secciones de funcionalidades extra implementadas, aspectos positivos de la resolución, desafíos encontrados y los casos de test.

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, -1)

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 FilaSatisfecha tomara el valor 1.

verificar_fila(+_, +_, +_, -0)

Este predicado se utiliza como caso base del predicado anterior, este predicado es útil cuando no se cumple que la fila satisfaga las pistas dadas, por lo tanto, FilaSatisfecha 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)

Este predicado 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, -1)

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

verificar_columna(+_,+_,+_,-0).

Este predicado se utiliza como caso base del predicado anterior, sirve para cuando no se cumple que la columna satisfaga las pistas dadas, por lo tanto, ColumnaSatisfecha 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)

Este predicado se utiliza como 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, +[], -[]).

Este predicado se utiliza como caso base número 1 del predicado anterior, si no hay más pistas, ni lista entonces se cumple que haya n casillas consecutivas pintadas y se retorna la lista vacía.

verificar_pconsecutivos(+0, +[X|Filarestante], -Filarestante)

Este predicado se utiliza como caso base número 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 ganó 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)

Este predicado se utiliza como caso sabe del predicado anterior, si no hay más 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)

Este predicado se utiliza como caso base 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).

Este predicado 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).

Este predicado se utiliza como caso base del predicado anterior, retorna la cantidad de elementos de una lista vacía que es 0.

comprobar_grilla_react(+Grilla, +PistasFilas, +PistasCol,

-FilaConPistas, -ColumnaConPistas)

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(+Grilla, +PistasCol, +CantColumnas,

-ColumnaConPistas)

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_columnas_react(+_,+ _, +0,-[]).

Este predicado se utiliza como caso base del predicado anterior, retorna una lista vacía cuando no hay por recorrer más columnas.

comprobar_todas_filas_react(+Grilla, +PistasFilas, +CantFilas,

-FilaConPistas)

Estrategia: Se comporta de manera análoga a comprobar_todas_columnas_react/4 pero con filas.

El predicado comprobar_todas_columnas_react/4 se utiliza para retornar la lista con las pistas que se cumplen en cada fila de la grilla.

comprobar_todas_filas_react(+_, +_, +0, -[]).

Este predicado se utiliza como caso base del predicado anterior, retorna una lista vacía cuando no hay por recorrer más filas.

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

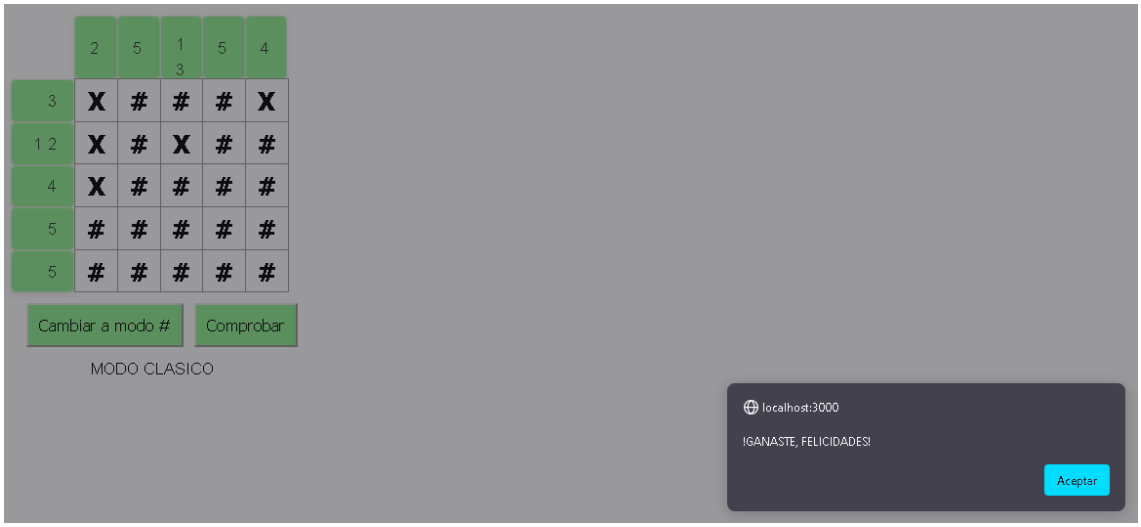
Uno de los principales desafíos fue el de no saber cómo empezar el proyecto, había mucho por hacer, varias carpetas, varios archivos de distintas extensiones, mucho código y al ser la primera vez que veíamos algo así se sintió bastante abrumador, una vez decididos sobre donde arrancar, que fue por el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

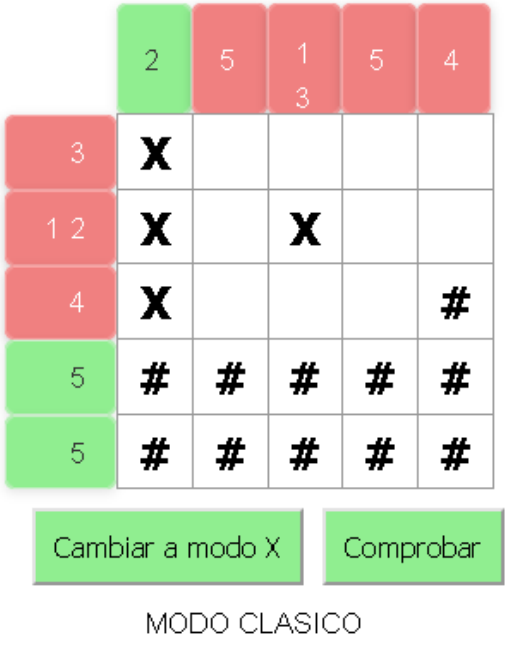
Programar en React también ha sido un desafío, como por ejemplo el cómo conectar si una pista está bien o mal con su color.

Casos de Test

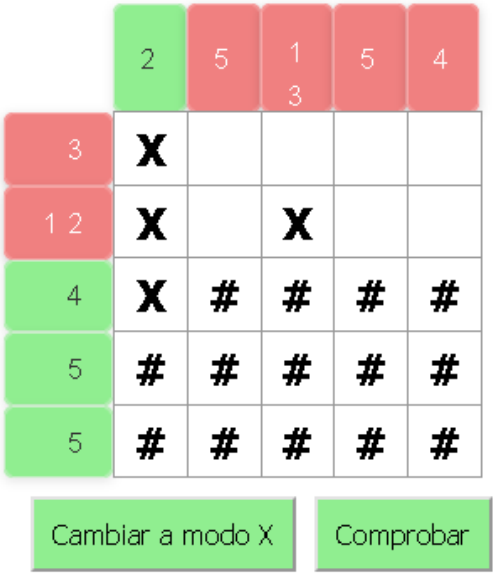
En caso de ganar el juego y presionar el botón “Comprobar” se chequea que haya ganado el juego y lo informa.



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

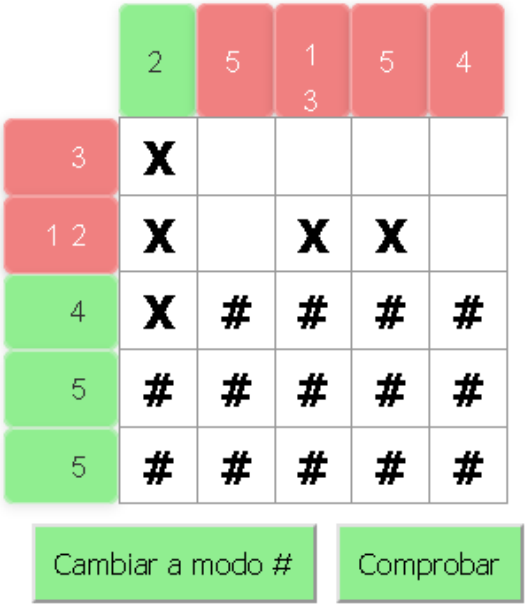


En caso de presionar el botón “Comprobar” y que el juego no esté terminado no sale ningún cartel y se podrá seguir jugando.



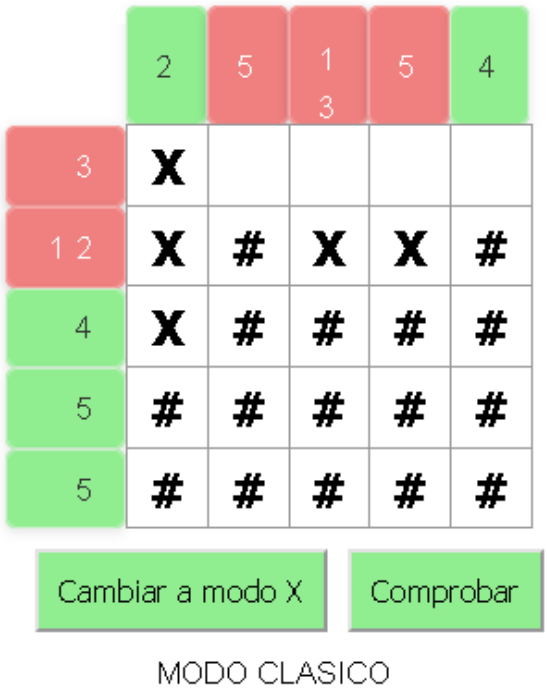
MODO CLASICO

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 #”.



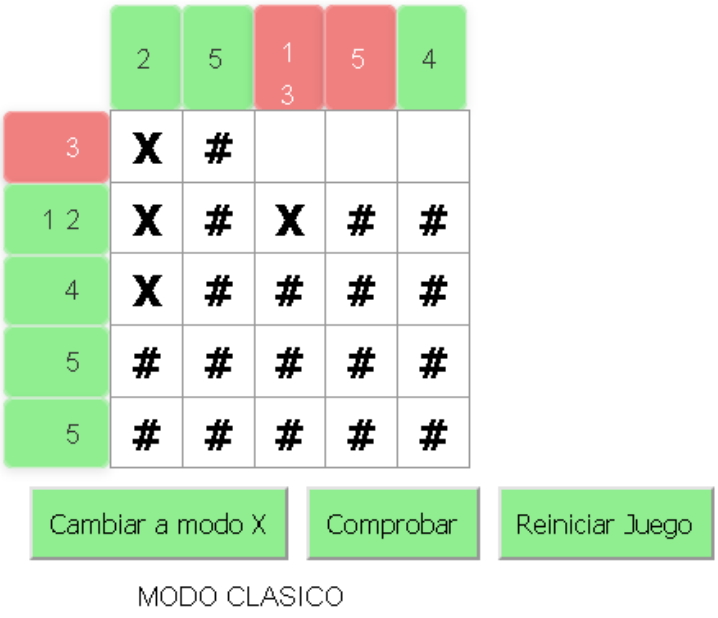
MODO CLASICO

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”.



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”: