DOCUMENTACIÓN DEL PROYECTO "NONOGRAMA"

"IMPLEMENTACIÓN EN REACT"

Alumnos

Latouquette, David Emanuel - López, David Estebán

Comisión N° 32

ÍNDICE

Introducción

En esta documentación se hablará sobre la implementación en React del proyecto “Nonograma”, haciendo énfasis en un alto nivel de los siguientes aspectos desarrollados en Prolog:

- Valores mantenidos en el estado del componente principal y su propósito.
- Consultas realizadas a Prolog
- ¿Cómo se actualiza el estado?

Finalmente, se tendrá una sección en la que se podrá observar una explicación breve de cómo se debe interactuar con la interfaz para hacer un correcto uso de la misma.

1. Componente Principal

La componente principal del proyecto es la clase ***Game***, en donde se realiza la consulta del programa sobre Prolog y se controla el estado del juego a partir de los valores que son asociados al tablero.

I. Propósito del componente principal

El propósito del componente ***Game*** es controlar el estado del juego. Esto lo realiza comprobando el estado de cada fila y columna y comprueba si se satisface la cantidad de celdas marcadas (de manera seguida) con las requeridas por la pista.

Este componente es el encargado de vincular la parte lógica implementada en Prolog. Dicha vinculación es realizada en forma de consultas (esto se explica en profundidad en otra sección), en donde se obtiene inicialmente el tablero y las pistas del juego.

II. Valores mantenidos en el componente principal

En el componente ***Game*** se mantienen ciertos valores que son necesarios para el correcto funcionamiento del programa.

▸ ***grid:*** Esta propiedad almacena la grilla del juego, que será el conjunto de celdas del tablero. Dicho conjunto se mantendrá constante durante toda la partida.

▸ ***rowClues:*** Esta propiedad almacena el conjunto de todas las pistas de las filas para un determinado tablero.

▸ ***colClues:*** Esta propiedad almacena el conjunto de todas las pistas de las columnas para un determinado tablero.

▸ ***waiting:*** Esta propiedad tiene como función almacenar un booleano con la función de verificar si se puede o no seguir modificando.

▸ ***opcion:*** Esta propiedad tiene como función almacenar el símbolo con el que se marca en el tablero. Dicha propiedad almacenará **#** (pinta una celda) o **X** (tacha una celda).

▸ ***satisfaccion_filas:*** Esta propiedad almacena una lista de valores 0 ó 1 que simbolizan la satisfacción de las pistas de las filas, donde satisfaccion_filas(i) almacena 0 si la fila i no satisface sus pistas, o 1 si la fila i satisface sus pistas.

▸ ***satisfaccion_cols:*** Esta propiedad almacena una lista de valores 0 ó 1 que simbolizan la satisfacción de las pistas de las columnas, donde satisfaccion_columnas(i) almacena 0 si la columna i no satisface sus pistas, o 1 si la columna i satisface sus pistas.

▸ ***victoria:*** Esta propiedad almacena un booleano, el cual inicialmente es ***false*** y solamente cambia de valor a ***true*** cuando se gana el juego.

III. Propósito de los valores en el componente principal

Todos los valores (mencionados en *"II. Valores mantenidos en el componente principal"*) tienen un propósito dentro del componente principal.

▸ El propósito de **grid** es almacenar el estado del tablero, pudiendo mantener la estructura del mismo. El tamaño del grid afectará significativamente a las pistas, puesto que la grilla debe tener un tamaño tal que toda pista pueda estar contenida dentro del tablero.

▸ El propósito de ***rowClues*** y ***colClues*** es mantener almacenadas al conjunto de pistas del juego. Puntualmente, indica cuántas marcas necesariamente debe haber en cada fila y en cada columna para que se satisfaga cada fila y cada columna.

▸ El propósito de ***waiting*** es permitir que se actualice el tablero mientras que las filas y columnas no satisfagan las pistas, caso que sucede cuando ***waiting*** es ***false***.

- Cuando dichas pistas son satisfechas, esta propiedad se setea en ***true***, haciendo que el usuario no pueda alterar el resultado del juego (esto es, cambiando el valor de una celda, lo que haría perder la condición de victoria).

▸ El propósito de ***opcion*** es mantener almacenado el símbolo con el que se marca en el tablero. Este símbolo se actualiza empleando los botones de selección, con los cuales se puede pasar de **#** (pintar una celda) a **X** (tachar una celda), y viceversa.

▸ El propósito de ***satisfaccion_filas*** y ***satisfaccion_cols*** es verificar las filas y columnas que están satisfechas de acuerdo a la comprobación de las marcas pintadas en el tablero.

- Estas propiedades son importantes, puesto que si ambos conjuntos contienen todas las filas y columnas respectivamente, esto se plasma en el programa como que se satisface todo el tablero, y por ende, termina el juego.

▸ El propósito de ***victoria*** es almacenar un booleano que indique si finalizó el juego, lo que ocurre cuando son satisfechas las columnas y las filas.

- Esta propiedad puntualmente permite mostrar un texto al usuario que le indica el estado del juego, esto es, si sigue en juego o si ganó.

2. Componentes secundarias

I. RadioButton.js

La clase ***RadioButton*** es una componente React que permite modelar un botón de selección y contendrá un valor definido por ***this.props.value***.

▸ **Propiedad "value":** Esta propiedad almacena contendrá un valor con el cual se inicializa el componente.

▸ **Propiedad "checked":** Esta propiedad almacena un booleano que indica si el botón está seleccionado o no.

- Este valor se obtiene por medio de ***this.props.checked.***

▸ **Propiedad "onChange":** Esta propiedad almacena un booleano que indica si se ha cambiado de selección, esto es, cuando se selecciona otro botón del mismo grupo pasando el valor al otro botón.

II. Board.js

La clase ***Board*** es un componente que modela al tablero/contenedor del proyecto, es decir, es donde estará contenido las celdas y los valores marcados en el juego.

Más precisamente, el ***Board*** contendrá $n * m$celdas que son modeladas por componentes ***Square***. Dichas componentes contendrán valores que serán contenidos en la clase ***Clue***.

III. Clue.js

La clase ***Clue*** es un componente que permite modelar las pistas que corresponden a las filas/columnas del tablero.

- Este componente es utilizado para establecer el texto de las pistas de cada fila y columna con sus respectivos números.

IV. Square.js

La clase ***Square*** es un componente que modela una celda del tablero, y está compuesta por un botón que mediante el evento ***onClick*** actualiza el valor del texto de dicho botón.

▸ **Propiedad value:** Esta propiedad es la que establece el texto que aparece en el botón contenido en el componente ***Square***.

3. Vinculación con Prolog

La implementación en React realiza una vinculación con Prolog en su mayoría, en forma de "consultas" en donde se realizan consultas para obtener el estado del tablero o se actualiza el tablero. A continuación se detallará con mayor precisión dichas consultas.

I. Consultas realizadas a Prolog

Al iniciar el programa, la primera consulta que se le realiza a Prolog es sobre el estado del tablero. En dicha consulta, se llama al predicado ***init/3*** de la parte lógica de Prolog y obtiene el tablero y las pistas de las filas y las columnas almacenados en dicho predicado.

Efectuada la consulta, y si es exitosa, se recupera la grilla, y las pistas correspondiente a las filas y las columnas y son almacenadas en las propiedades grid, rowClues y colClues respectivamente.

Cargados dichos datos, es cargado el tablero en ***Game*** y es posible efectuar cualquier operación de actualización sobre el tablero (ver "*II. Actualizaciones realizadas a Prolog*").

II. Actualizaciones realizadas a Prolog

Luego de la consulta anterior, toda acción que se realice sobre el tablero repercutirá en realizar una consulta de actualización sobre Prolog.

La actualización del estado del juego se realiza empleando el predicado ***put/8***, donde se actualiza solamente la celda modificada, se comprueba que filas y columnas son satisfechas y se plasma en el tablero.

Realizada con éxito la consulta de actualización, se recupera el tablero modificado y se lo almacena en la propiedad grid del componente ***Game***. A su vez, también es recuperado el conjunto de filas y columnas que son satisfechas.

Recuperados estos dos últimos conjuntos (filas y columnas), se comprueba si están todas satisfechas. En caso de un resultado afirmativo, el juego finaliza y se setea a victoria con ***true***.

3. ¿Cómo se actualiza el estado?

El estado del tablero se actualiza empleando las consultas de actualización que se tiene implementadas en la parte lógica de Prolog.

También, el componente ***Game*** almacena propiedades que determinan el estado del juego. El estado del juego depende de los valores mencionados anteriormente (puntualmente en "1. Componente Principal").

Precisamente, el proyecto captura los eventos producidos por el usuario y va alterando su estado de acuerdo al tipo de evento:

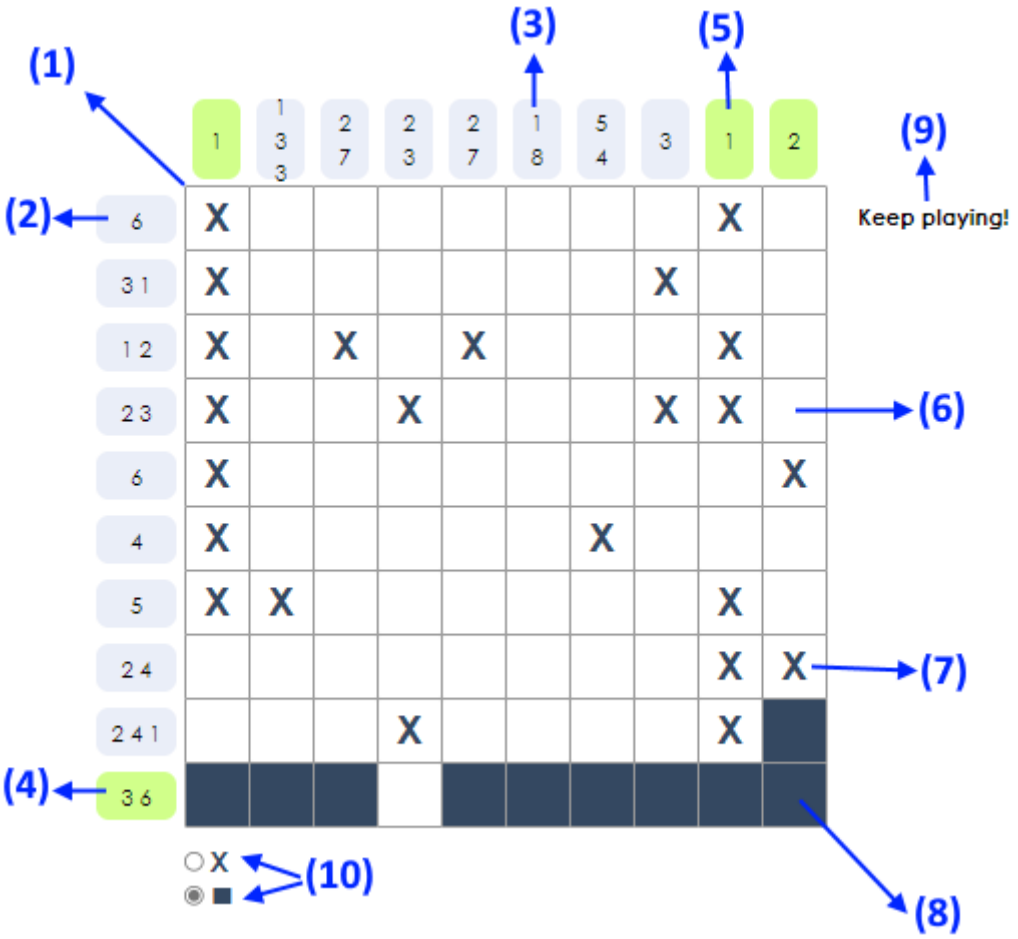
1). Cuando se efectúa el evento ***onChange*** en los botones de selección, esto desencadena que se produzca un cambio de estado en la propiedad ***opcion***, actualizando su valor al del botón de selección seleccionado.

2). Cuando se efectúa un evento ***click*** sobre las celdas del tablero, esto repercute en una actualización del tablero, en donde se actualiza dicha celda de alguna de las formas mencionadas en "4. Interacción con la interfaz - II. Pasos requeridos para la interacción con la interfaz".

Si dicha actualización desencadena en que todas las pistas son satisfechas, entonces se tiene que el juego finalizó, ganando el juego, cambiando de estado a la propiedad victoria a true y deteniendo el juego para que no se cambie el estado.

4. Interacción con la interfaz

I. Interfaz del proyecto



Referencias

(1). Tablero/grilla.
(2). Pista de la columna.
(3). Pista de la fila.
(4). Pista de la columna satisfecha.
(5). Pista de la fila satisfecha.
(6). Celda sin marcar (celda vacía).
(7). Celda marcada (celda tachada).
(8). Celda marcada (celda pintada).
(9). Texto.
(10). Botones de selección.

II. Pasos requeridos para la interacción con la interfaz

Paso 1: Selección de marcado

Para comenzar, es necesario seleccionar entre pintar o tachar una celda. La selección de marcado se realiza seleccionando uno de los botones de selección (10). La opción que esté seleccionada será con la que se marque en el tablero.

Paso 2: Marcado/desmarcado

El marcado/desmarcado se realiza sobre cualquier celda del tablero. Los marcados de celdas pueden ser pisados o incluso borrados, por lo tanto, hay distintos casos que pueden ocurrir:

1). Opción seleccionada: Pintar celda (8)

- **I.** Si la celda está vacía (6), entonces se pinta (8).
- **II.** Si la celda también está pintada (8), entonces se desmarca y queda vacía (6).
- **III.** Si la celda está tachada (7), entonces la celda se pinta (8).

2). Opción seleccionada: Tachar celda (7).

- **IV.** Si la celda está vacía (6), entonces se tacha (7).
- **V.** Si la celda también está tachada (7), entonces se desmarca y queda vacía (6).
- **VI.** Si la celda está pintada (8), entonces la celda se tacha (7).

Paso 3: Satisfaciendo filas/columnas

Para satisfacer una fila/columna se debe prestar atención a las pistas **(2)** y **(3),** cuando se satisface la cantidad de celdas pintadas consecutivamente con respecto a la pista de la columna o fila, entonces la pista es satisfecha y aparece de color verde según sea, es decir, la pista de la columna (4) o de la fila (5) aparece en color verde.

Puede ocurrir que una pista contenga más de un número, por lo tanto, para satisfacer dicha pista se debe pintar de manera de satisfacer la cantidad indicada por la pista y separar cada número con un espacio como mínimo. Esto se puede observar en la última fila de la figura de referencia dada.

Paso 4: Juego ganado

Una vez que se cumpla que todas las filas y todas las columnas estén satisfechas, esto es, cuando las columnas y las filas tienen sus pistas todas en verde, el proyecto actualiza el texto (9) con la leyenda “Has ganado!”.

Finalizado el juego, el juego realizado no puede ser alterado, por lo tanto, si se desea utilizar el juego nuevamente, se deberá recargar la página.