Proyecto 1 - Lógica para Ciencias De La Computación

Informe de alto nivel acerca de la implementación

Comisión Nro 18
GATTI, TOBIAS – REALE, GUIDO

Vinculación entre React y el servidor Prolog

Al iniciar la aplicación, se llama a “PengineClient.init(onServerReady)” para inicializar la conexión con el servidor Prolog. Cuando el servidor está listo, la función “onServerReady” se llama y configura las variables de estado “Grid” y “numOfColumns” para almacenar los datos iniciales del juego.

- El usuario interactúa con el juego dibujando un camino en una grilla(tablero), dicho camino está representado por la variable de estado “path”. Cuando el usuario completa el camino, se construye una consulta de Prolog que se envía al servidor con “pengine.query()”. La respuesta del servidor se procesa en “onPathDone()”, actualizando el estado del juego y animando los cambios en la grilla.
- El botón de “Colapsar Iguales” también envía una consulta al servidor Prolog, actualizando el estado del juego y animando los cambios en la grilla.

En resumen, este código React se comunica con un servidor Prolog a través de “PengineClient” y actualiza el estado del juego en función de las respuestas del servidor y las interacciones del usuario.

Aspectos a tener en cuenta

Se utilizó la clase click.mp3 para agregar la funcionalidad del sonido cada vez que se clickea en un bloque del tablero, puede llegar a haber un posible problema de compatibilidad, en ese caso quitar esa porción de código para evitar el problema.

Manejo de las consultas en el servidor

Al servidor Prolog le llegan dos tipos de consultas desde React, como fue mencionado anteriormente;

1) la primera de ellas es la consulta que se desprende del comportamiento que realiza el usuario del juego cuando dibuja un camino en el tablero, y lo suelta para que este se elimine y se reemplace el ultimo cuadrado seleccionado por la potencia correspondiente. Cuando el usuario realiza esta acción, desde la clase game.js se construye una consulta que llama al predicado Prolog join/4

```
join(Grid, _NumOfColumns, _Paxth, RGrids):-
    sumarPath(Grid, _Paxth, _NumOfColumns, Suma),
    menorPotenciaDe2(Suma, Potencia),
    eliminarLista(Grid,_Paxth,_NumOfColumns,Potencia,Resultante),
    eliminarUltimo(_Paxth,PathAux),
    ordenarPorX(_Paxth,PathAux2),
    gravedad(Resultante,PathAux2,_NumOfColumns,Res),
    reemplazarCeros(Res,Res2),
    RGrids = [Resultante,Res,Res2].
        % formula para transformar una coordenada en indice--> (X * NumOfColumns) + (Y mod NumOfColumns)
```

Con los siguientes parámetros:

- Grid = lista de elementos que representa el tablero del juego.
- _NumOfColumns = número de columnas del tablero.
- _Pathx = lista de listas, donde cada SubLista es una coordenada del camino a eliminar.
- RGrids = lista de Grids resultante que se va a enviar como respuesta a la consulta (se va a utilizar para animar los movimientos en el tablero).

A grandes rasgos, lo que se realiza en este predicado es lo siguiente:

En primera instancia, se suman los números que forman parte del camino generado por el usuario(_Paxth) mediante el predicado sumarPath/4, luego, el resultado obtenido se utiliza para calcular la menor potencia de dos menor o igual a dicha suma mediante el predicado menorPotenciaDe2/2.

Llegado este momento, se llama al predicado eliminarLista/5, que se encarga de ir descomponiendo la lista almacenada en la variable _Paxth reemplazando los valores en

cada posición por 0, y la última posición por la potencia correspondiente calculada en el paso anterior.

Un ejemplo de esto es:

Suponiendo que tenemos el tablero con la siguiente disposición:

Enviando al predicado eliminarLista/5 la Grid actual, y la lista que representa al Path marcado en pantalla → [[4,4],[4,3],[3,3]], dicho predicado devolvería

la lista que representa a la siguiente disposición del tablero(almacenado en la variable Resultante):

Aquí, una vez ejecutado el predicado eliminarLista/5, se llama al predicado eliminarUltimo/2 que elimina el último elemento del camino creado por el usuario (única coordenada perteneciente a la lista que tenga un numero distinto de 0 en su contenido en este momento, y por lo tanto no nos sirve para la aplicación de la gravedad), y a esa lista resultante, sin el último elemento, la ordenamos de manera descendente con respecto a la componente x mediante el predicado ordenarPorX/2.

Siguiendo con el ejemplo dado anteriormente, luego de ejecutar el primer predicado de estos, la nueva lista seria [[4,4],[4,3]], y con la ejecución del segundo, en este caso quedaría igual.

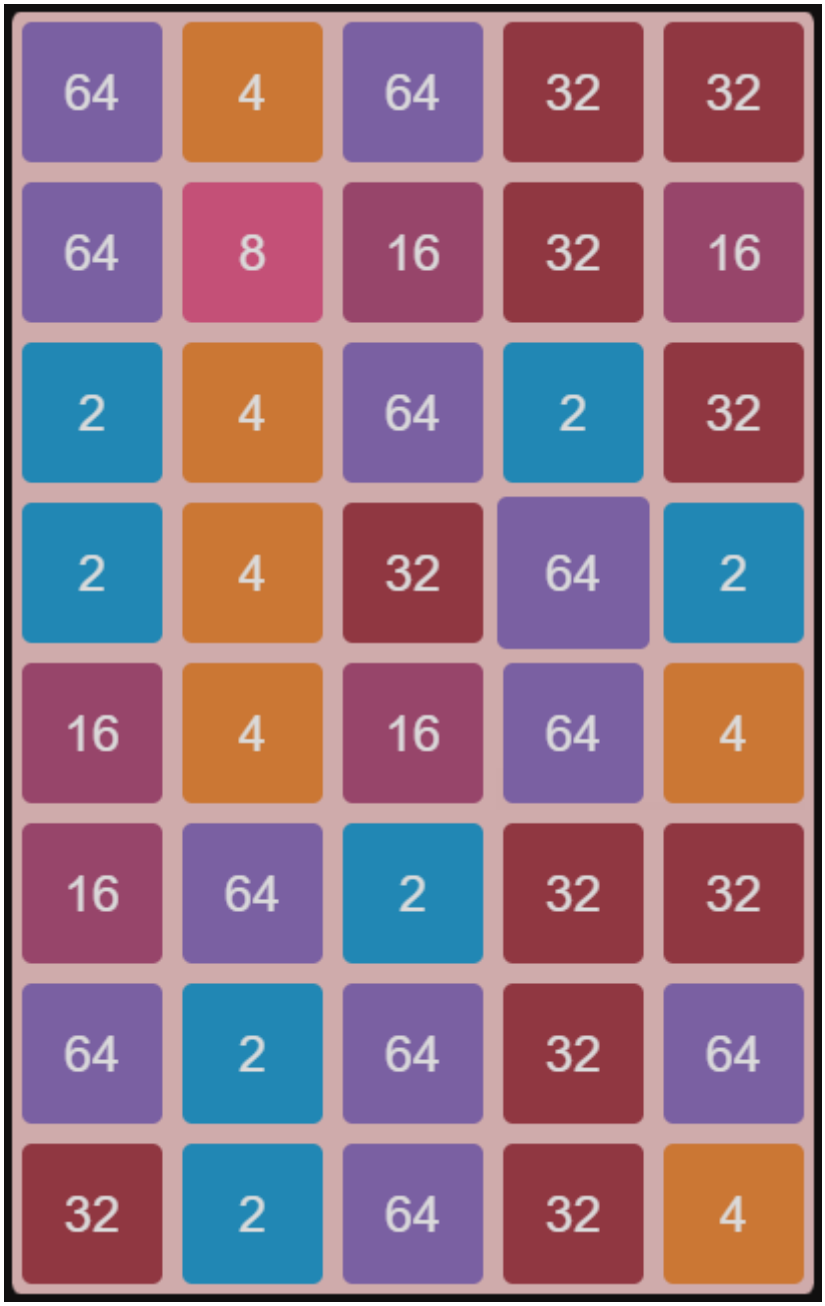
Luego, el resultado parcial que nos devolvió ordenarPorX/2 lo enviamos en la llamada al predicado gravedad/4, que se va a encargar de devolver una nueva Grid “afectada” por la gravedad respetando el comportamiento solicitado por la cátedra.

Continuando nuestro ejemplo, aquí el predicado gravedad nos devolvería la lista que representaría a la siguiente disposición del tablero (almacenado en la variable Res):

Como anteúltimo paso, al resultado parcial que nos devolvió el predicado gravedad/4(es decir, la Grid con la gravedad aplicada y todos los ceros en su posición

final) lo pasamos como parámetro al predicado reemplazarCeros/2 que se encargará de reemplazar ceros por potencias de 2 aleatorias.

Terminando el ejemplo en este paso, luego de ejecutar el predicado reemplazarCeros/2, el mismo nos devolvería como resultado una lista que representa la siguiente disposición del tablero (almacenada en la variable Res2):



Luego, como paso final, se almacena en la variable RGrids(que es la que se va a devolver), las Grids parciales que fuimos generando a lo largo del predicado, esto es:

- Resultante: Grid con el camino eliminado y la última posición reemplazada por la potencia correspondiente.
- Res: Grid con la gravedad aplicada, los ceros lo más arriba posible.
- Res2: Grid con los ceros reemplazados por potencias de 2 aleatorias.

2) La segunda consulta que se envía desde React hacia el servidor Prolog, se desprende de la acción de apretar el botón “Colapsar Iguales” ejecutada por el usuario. Cuando esto sucede, desde la clase game.js se construye una consulta Prolog que llama al predicado booster/3

```
/*  se encarga de realizar el comportamiento esperado al presionar el boton colapsar iguales
*   colapsa todos los grupos de bloques adyacentes y de igual valor
*   reemplaza a la posición más abajo y a la derecha del grupo por la potencia
*   correspondientes de acuerdo a la sumatoria de los bloques de ese grupo.
*   luego aplica gravedad. y reemplaza los 0s por potencias de 2 aleatorias.
*/
booster(Grid,NumOfColumns,RGrids):-
    length(Grid, Size),
    CantidadFilas is Size/NumOfColumns,
    ListaGrupos = [],
    Indice = 0,
    agregarAListaGrupos(Grid,NumOfColumns,ListaGrupos,CantidadFilas,Indice,ResAux),
    eliminarListasVacias(ResAux,ResAux2),
    flatten(ResAux2,ListaGruposAplanada),
    eliminarBooster(Grid,ResAux2,NumOfColumns,ResFinal),
    traducirIndicesACoordenadas(ListaGruposAplanada,[],NumOfColumns,ListaCoordenadasGrupos),
    gravedad(ResFinal,ListaCoordenadasGrupos,NumOfColumns,Res),
    reemplazarCeros(Res,Res2),
    RGrids = [ResFinal,Res,Res2].
```

Con los siguientes parámetros:

- Grid = tablero actual del juego.
- NumOfColumns = número de columnas del tablero.
- RGrids = lista de tableros resultantes.

En primera instancia se llama a la variable predefinida en Prolog lenght/2 para calcular el largo de la Grid que representa el tablero, y usar este resultado junto con el numero de columnas para calcular la cantidad de filas que tiene el tablero. Se crea una lista vacía, se iguala la variable Índice a 0 (para arrancar recorriendo la Grid desde la primera posición).

Luego, se llama al predicado agregarAListaGrupos/6, pasándole como parámetro la Grid actual, el número de columnas, la lista vacía previamente creada, el índice, la cantidad de filas, y la variable donde se va a almacenar la lista resultante. Este predicado se va a encargar de devolver una lista de listas, donde cada sublista contiene los índices del grupo correspondiente de bloques adyacentes de igual valor, cuando un

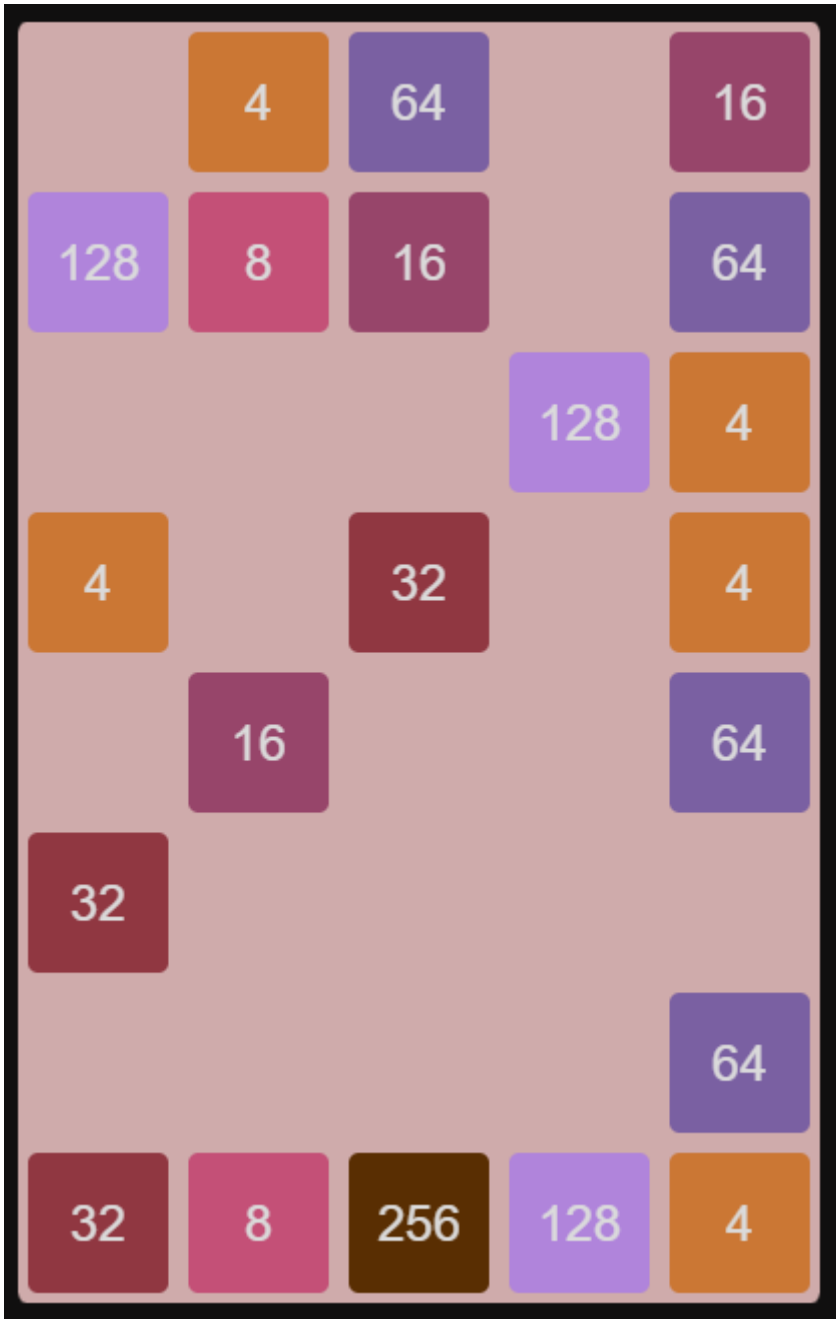
bloque no tiene adyacentes de igual valor se agrega una sublista vacía como representación.

En el siguiente paso, se llama al predicado eliminarListasVacias/2 con el resultado del paso anterior, para eliminar las listas vacías que representan a los bloques que no tienen adyacentes de igual valor.

Tomando como ejemplo la Grid inicial que tomamos en el ejemplo 1:

Llamamos al predicado eliminarBooster/4 con la lista resultante, que se va a encargar de ir tomando cada sublista de esa lista (que cada una representa un grupo) y eliminándolas, reemplazando la posición mas a la derecha por la potencia correspondiente.

Luego de ejecutar este predicado, el mismo nos devolvería una lista que representaría el siguiente estado del tablero (almacenado en la variable ResFinal):

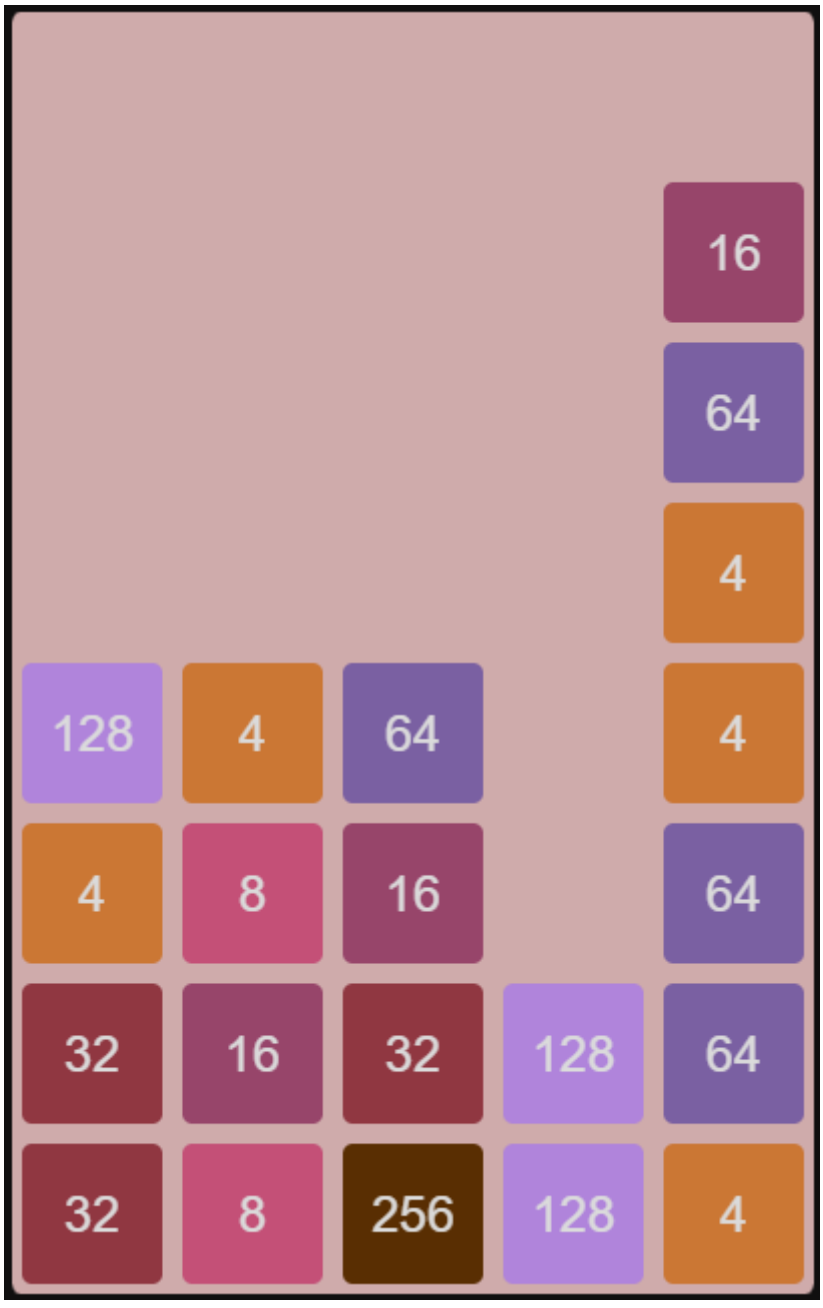


El predicado predefinido Prolog flatten/2 recibe una lista de sublistas, y la aplana, es decir, devuelve una lista general con todos los elementos de las sublistas.

Aquí, llamamos al predicado traducirIndicesACoordenadas/4 con la lista de grupos aplanada por el método anterior, que va a traducir cada coordenada de la lista a su índice correspondiente.

Luego, se realiza un manejo idéntico al que se realiza con el Path recibido en el Join/4 mediante los predicados gravedad/4 y reemplazarCeros/2.

Luego de la ejecución del predicado gravedad/4, a partir de la ultima Grid mostrada, dicho predicado devolverá una lista que representa el siguiente estado del tablero(almacenada en la variable Res):



Y luego de ejecutar el predicado reemplazarCeros/2, el resultado seria una lista que representaría el siguiente estado del tablero (almacenada en la variable Res2):

El paso final consiste en almacenar en la variable RGrids(que es la que se va a devolver), las Grids parciales que fuimos generando a lo largo del predicado, esto es:

- ResFinal: Grid con los grupos eliminados y la posición más abajo a la derecha de cada uno reemplazada por la potencia correspondiente.
- Res: Grid con la gravedad aplicada, los ceros lo más arriba posible.
- Res2: Grid con los ceros reemplazados por potencias de 2 aleatorias.

Aspectos A Mejorar

Cuando desde el predicado agregarAGrupoAux/7 llamamos al predicado chequearCaso/9, dicho predicado consta de 12 casos(9 para cada caso posible del tablero actual, y 2 para el caso en que se modifique la grilla y se tenga un tablero de una sola fila), que fue la decisión inicial que tuvimos para modelarlo, pero podrían haberse tenido en cuenta buenas prácticas de programación y modularizar la generación de los índices adyacentes mediante un predicado auxiliar, que no dependa del caso particular en el que se está, y solo devuelva una lista de índices adyacentes que caigan dentro del tablero que se tiene, para luego seguir el manejo de esos índices como el predicado principal lo requiera. De esa manera no tendríamos que tener tanta reescritura o repetición de código, quedando éste mucho más prolijo y ordenado.