Proyecto 1: SUDOKU

Lógica en Cs. De la Computación

15/05/2017

Matías Massetti y Germán A. Gómez

Índice

Definiciones y especificaciones de requerimientos

Definición general

Se debe realizar el juego llamado sudoku para la materia Lógica en Ciencias de la Computación. Su objetivo es rellenar una cuadrícula de 9 × 9 celdas (81 casillas) dividida en subcuadrículas de 3×3 (también llamadas "cajas" o "regiones") con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. La única restricción es que no debe repetirse ninguna cifra en una misma fila, columna o subcuadrícula.

Especificaciones

Implementar el Sudoku con los botones Comprobar, el cual verificará si la configuración de ese momento en el tablero tiene una solución, y Resolver, el cual mostrará un resultado de dicha configuración en caso de existir.

Tiene como objetivo programar el juego en dos partes:

- La interfaz gráfica deberá ser llevada a cabo en JAVA, y
- La lógica de las funcionalidades 'Comprobar' y 'resolver', deberá ser implementada en PROLOG, al igual que la verificación de la validez de una movida. Es importante tener en cuenta que se requiere no usar librerías y predicados predefinidos, a excepción de los vistos hasta la fecha en el cursado de Lógica en Ciencias de la Computación del actual año.

El proyecto cuenta con cuatro configuraciones:

- *Cargar números manualmente:* consiste en que el usuario pueda ingresar su propio tablero para luego jugar la partida, y
- *Tablero 1:* consiste en un tablero predeterminado con una dificultad sencilla.
- *Tablero 2:* consiste en un tablero predeterminado proporcionado por la cátedra. Además posee una dificultad media.
- *Tablero 3:* consiste en un tablero predeterminado. Tiene una dificultad extremadamente complicada, por lo cual el programa puede llegar a tardar 10 segundos en encontrar un resultado.

Procedimiento de instalación y prueba

Los pasos para utilizar el juego son:

1. Tener instalado en la PC: JAVA, SWI PROLOG, JDK. Además es necesario agregar a la variable de entorno `path´ la ruta del directorio bin de la instalación del swi-prolog (generalmente c:\Program Files\swipl\bin), y luego reiniciar, para que funcione.
2. Ejecutar con Java el archivo '*sudoku.jar*', teniendo en el mismo directorio a los archivos jpl.jar (dentro de la carpeta bin) y lógica.pl.
3. En la pantalla principal apretar la opción '*Comenzar Juego'.*

4. Elegir la configuración que se desea (Cargar números manualmente, Tablero 1, Tablero 2 o Tablero 3).
5. Para ingresar números hay dos opciones: la primera es hacer clic en la casilla e ingresar el número a través del uso del teclado de usuario, y la segunda forma es, realizar un clic en la casilla y apretar en la parte superior en el número que se quiere ingresar.
6. A medida que transcurre la partida se puede comprobar si hay un resultado con los números ingresados, resolver el sudoku en caso de que haya una posibilidad, reiniciar el tablero a la configuración inicial y volver al menú de inicio.

El juego fue realizado en SWI Prolog y en Eclipse Neón desde el sistema operativo Windows 10.

Arquitectura del Sistema

Descripción

El proyecto se divide en dos partes:

- *Se* dividió en tres paquetes las responsabilidades que tiene Java. El primer paquete maneja la sección gráfica mediante los archivos VentanaInicial.java y GUI.java. Ninguno de ellos tiene control sobre la lógica de PROLOG directamente. El segundo paquete contiene las imágenes que tiene el juego. Por último, el tercer paquete, contiene el archivo lógica.java, el cual se encarga de hacer la conexión JAVA-PROLOG; todo tipo de consulta que realice el programa lo hace a través de este utilizando lógica.pl.
- La segunda parte consiste en la parte lógica hecha en PROLOG la cual se encuentra en 'logica.pl'.

Descripción Detallada

JAVA:

- VenatanaInicial.java: contiene parte gráfica de la ventana al inicial del juego.
- GUI.java: contiene la parte gráfica del menú y del manejo del tablero.
- Logica.java: contiene todo tipo de consulta que debe hacerle GUI u otra clase que necesite hacer consultas a lógica.pl

PROLOG

Dentro del archivo se guarda cada posición del tablero que posee un número asignado mediante el uso de hechos, casilla(N, F, C), donde N es el número de la Fila F y la columna C. Por lo cual a medida que el usuario ingresa números, el programa va agregando hechos de forma tal que el tablero se guarda en su totalidad dentro del archivo PROLOG. Además se tiene predicados que resuelven, comprueban la validez de filas, columnas y regiones, insertan y eliminan números en el tablero y que reinician el tablero.

La estrategia utilizada para resolver el Sudoku es la del Backtracking. El programa empezará en la casilla [1,1] insertando números válidos (del 1 al 9), hasta finalizar en la casilla [9,9]. En caso de encontrarse con una casilla donde no es posible agregar ningún número, retrocede a la anterior, prueba un nuevo número en ella y prosigue. Si la ejecución llega a probar todas las opciones la casilla [1,1] o la primera que no sea predeterminada, se entenderá que no hay solución posible para ese tablero.

Dependencias externas

Se necesitan tener junto al archivo ejecutable sudoku.jar, la librería jpl.jar y lógica.pl.

Documentación

Lógica.pl

- Sudoku(-L): este predicado nos permite recibir una Lista L que contiene 9 listas, cada una de ellas representa una fila del tablero. El número 0 (cero) representa una posición en la que no hay un valor insertado.
- resolver(): este predicado resuelve el tablero con la configuración actual del tablero. Si es posible encontrar el resultado, lo guarda en los hechos casillas(N, F, C). En caso de no haber solución, devolverá falso. Para pedir el tablero resuelto se recomienda usar el predicado 'sudoku(L)'.
- buscar_numero(?N,+F,+C): busca un número válido para la casilla que se encuentra en la fila F y columna C. También sirve para validar si un valor N es válido en la casilla de la fila F y columna C.
- validar_fila(+N,+F): retorna si es válido insertar el número N en la fila F.
- validar_columna(+N,+C): retorna si es válido insertar el número N en la columna C.
- validar_bloque(+N,+F,+C): retorna si es válido insertar el número N en la región de la casilla que se encuentra en la fila F y columna C.
- insertar(+V,+F,+C): inserta el valor V en la fila F y columna C. Utiliza retractall para eliminar cualquier punto que esté en la posición, asserta para insertar el hecho y retract y fail para eliminar un hecho al hacer backtracking.
- Eliminar(+F,+C): elimina el hecho que se encuentra en la fila F y columna C.
- Eliminar_todo(): elimina todos los hechos que representas posiciones con números que se encuentran en el tablero.

Código PROLOG

```
%casilla(?Valor,?F,?C)
%Representa una posición del tablero con su respectivo valor.
:-dynamic casilla/3.


%num(N)
%numeros válidos del tablero
num(1).
num(2).
num(3).
num(4).
num(5).
num(6).
num(7).
num(8).
num(9).


%sudoku(-L).
%Retorna una lista L que contiene 9 listas, cada una correspondiente a un
bloque.
sudoku(L):- sud_aux(1,1,[],L).


%sud_aux(+F,+C,+Aux,-L).
% Retorna una lista L que contiene 9 listas, cada una correspondiente a un
fila.
% Aux es una lista auxiliar donde se van guardando las filas.
sud_aux(10,1,[],[]). %Termino de leer el tablero
sud_aux(F,10,Aux,[Aux|L]):- F2 is F+1, sud_aux(F2,1,[],L). %Termino una fila.
sud_aux(F,C,Aux,L):- casilla(N,F,C), C2 is C+1, append(Aux,[N],Aux2),
sud_aux(F,C2,Aux2,L). % Guardamos el punto
sud_aux(F,C,Aux,L):- not(casilla(_,F,C)), C2 is C+1, append(Aux,[0],Aux2),
sud_aux(F,C2,Aux2,L). % No existe el punto


%resolver/0
%Resuelve el sudoku
resolver():- res(1,1).


%res(+F,+C)
%Resuelve el sudoku desde la casilla (F,C) hasta la casilla (9,9)
res(9,10).
res(F,10):- F2 is F+1, res(F2,1).
res(F,C):- not(casilla(_,F,C)), buscar_numero(N,F,C), insertar(N,F,C), C2 is
C+1, res(F,C2).
res(F,C):- casilla(_,F,C), C2 is C+1, res(F,C2).


%buscar_numero(?N,+F,+C)
%Busca un numero válido para la casilla (F,C).
%F y C son numeros entre 1 y 9.
buscar_numero(N,F,C):- num(N), not(casilla(N,_,C)), not(casilla(N,F,_)),
validar_bloque(N,F,C).
```

```
%validar_fila(+N,+F)
%Retorna true si es válido insertar el número N en la Fila F.
%F pertenece al intervalo [1,9].
validar_fila(N,F):- not(casilla(N,F,_)).


%validar_columna(+N,+C)
%Retorna true si es válido insertar el número N en la columna C.
%C pertenece al intervalo [1,9].
validar_columna(N,C):- not(casilla(N,_,C)).


%validar_bloque(+N,+F,+C)
%Retorna true si es válido insertar el número N en la region de la casilla
(F,C).
%F y C pertenecen al intervalo [1,9].
validar_bloque(N,F,C):- FMax is ((((F-1)//3)+1)*3), CMax is ((((C-
1)//3)+1)*3), not(casilla(N,FMax,CMax)), C1 is CMax-1,
not(casilla(N,FMax,C1)), C2 is CMax-2, not(casilla(N,FMax,C2)), F1 is FMax-1,
not(casilla(N,F1,CMax)), not(casilla(N,F1,C1)), not(casilla(N,F1,C2)), F2 is
FMax-2, not(casilla(N,F2,CMax)), not(casilla(N,F2,C1)),not(casilla(N,F2,C2)).


%insertar(+V,+F,+C)
%Agrega el número V en la casilla (F,C).
%Primero elimina todos los valores que tiene asignado la posición (F,C), para
luego insertar.
insertar(V,F,C):- retractall(casilla(_,F,C)), asserta_2(casilla(V,F,C)).


%asserta_2(+casilla(V,F,C))
%Agrega el hecho casilla(V,F,C).
asserta_2(casilla(V,F,C)):- asserta(casilla(V,F,C)).
asserta_2(casilla(V,F,C)):- retract(casilla(V,F,C)), fail.


%eliminar(+F,+C)
%retorna true si elimina todos los valores que corresponden a la casilla
(F,C)
%F es la fila y C la columna. F y C pertenecen al intervalo [1,9].
eliminar(F,C):- retractall(casilla(_,F,C)).


%eliminarTodo()
%retorna true si elimina todas las casillas del juego.
eliminarTodo():- retractall(casilla(_,_,_)).
```

Aspectos relevantes

Decisiones adoptadas y conclusiones

- Se optó que el usuario pueda inserta números por teclado o por botones.
- Se decidió usar hechos ya que su funcionamiento nos pareció mucho más sencillo que el uso de cualquier tipo de listas.
- El predicado validar_bloque(N, F, C) quedo menos 'elegante' a cambio de bajar su tiempo de resolución. Anteriormente se usaba el predicado predefinido findall para obtener las casillas, para luego verificar si N estaba en alguna de ellas a través de un pertenece. Pero como era costoso su tiempo, se prefirió calcular a través de F y C una posición del bloque y a partir de ella ir consultando hecho a hecho si pertenece N a ellos. Este aspecto bajo notablemente las inferencias a la mitad.
- Se priorizó buscar de manera más veloz un resultado del tablero que tiene solución, antes que encontrar que no tiene solución perjudicando los casos en que si tenía. La idea era crear un predicado llamado probabilidades(-L), el cual se encargaba de retornar una lista L que contenía listas, cada una de ellas correspondiendo a los números válidos que tenia cada casilla. Esto nos permitía saber antes de insertar valores si había una casilla vacía que no tenía números posibles. El problema consistía que si el tablero tenia solución o todas las casillas tenían un posible valor, se incrementaba el tiempo de resolución.
- El modo cargar manualmente lo que hace es cargar un tablero vacio para que el usuario ingrese los números, pero como este tablero tiene las mismas propiedades (porque es el mismo) que el tablero de cuando se está jugando, chequea la validez de la fila, columna y bloque. Al apretar el botón de cargar el tablero, lo que realiza el juego es poner como no editables esos lugares y agregar los botones de comprobar y reiniciar. Este hecho nos ahorra mucho código y le posibilita al usuario insertar al menos un tablero válido a nivel de filas, columnas y regiones, teniendo cero, una o más soluciones.
- Para facilitar la tarea, utilizamos el interprete de PROLOG que se encuentra en http://swish.swi-prolog.org/ y el programa NOTEPAD++.
- En lógica.java, en el método resolver, se tuvo que agregar y 'embarrar' el código para solucionar un problema que tuvimos con la librería jpl.jar. Lo que sucedía era que al pedir resolver a PROLOG desde JAVA, el tiempo de ejecución iba aumentando ejecución tras ejecución, lo cual hacia que un tablero que se tardaba 0,7 segundos en ser resuelto, pasaba a tardar 3 segundos, 8 segundos y así sucesivamente en aumento. Para corregir esto, lo que hicimos es reiniciar el archivo lógica.pl en el método resolver. La idea es hacer un resguardo del tablero en JAVA, recargar lógica.pl, insertar el tablero guardado en JAVA en PROLOG y recién en esta instancia, pedir el resolver. Este detalle hizo que el tiempo de resolver tarde un poco más pero que siempre sea el mismo tiempo.
- Si el usuario opta por ingresar el número por teclado, deberá apretar enter para validar la jugada, en caso contrario no se efectuará la jugada en la lógica.

- Notamos que en las computadoras donde no se agrega a la variable de entorno `path´ la ruta del directorio bin de la instalación del swi-prolog no funciona el programa.

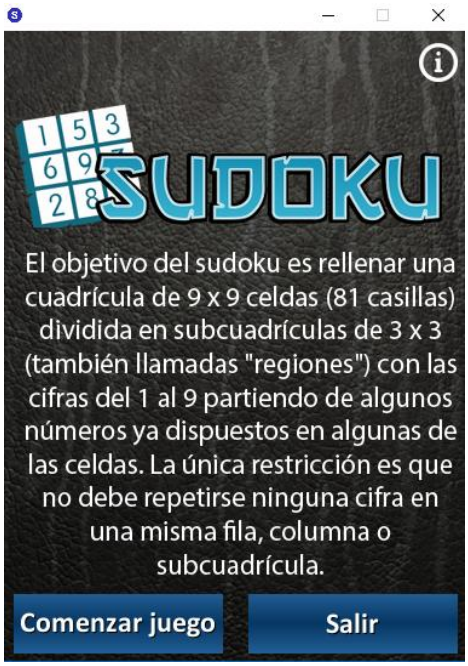
Manual de Usuario

Objetivo del juego

El objetivo del sudoku es rellenar una cuadrícula de 9 x 9 celdas (81 casillas) dividida en subcuadrículas de 3 x 3 (también llamadas "regiones") con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas de las celdas. La única restricción es que no debe repetirse ninguna cifra en una misma fila, columna o subcuadrícula.

Pantalla inicial

Al iniciar el juego la primera pantalla que se visualiza es la siguiente:



Aquí se brinda una breve descripción del juego e información relacionada. Para dar inicio al juego se debe seleccionar la opción **Comenzar juego**

Selección modo de juego

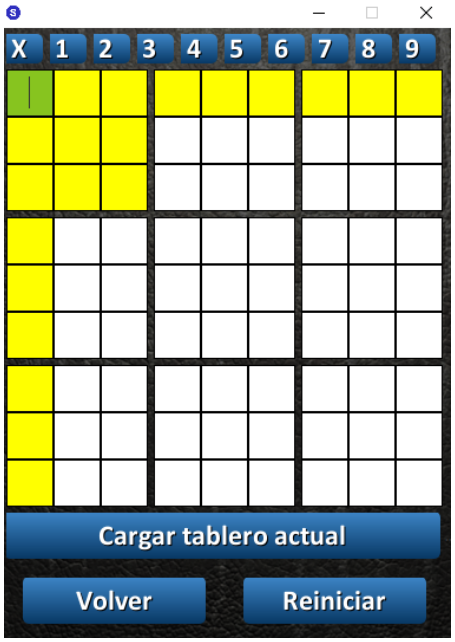
Para dar comienzo al juego se deberá elegir una de las opciones siguientes:

La primera opción es **Cargar números manualmente** y permite cargar una configuración disponiendo cifra por cifra en el tablero el número deseado.

Las opciones restantes son **tableros predeterminados** que varían en su dificultad en fácil, media o difícil.

Juego

Opción 1: Cargar números manualmente



Si se elige esta opción se deberá cargar el tablero con los valores deseados y cuando se haya finalizado de cargar se debe seleccionar la opción **Cargar tablero actual.**

Además se puede elegir **volver** al menú anterior o **reiniciar** el tablero.

Para cargar los números se puede utilizar la botonera de la parte superior de la pantalla, que además permite borrar el valor de una casilla, o bien utilizar el teclado.

IMPORTANTE: Si se utiliza el teclado se debe confirmar el valor ingresado presionando **ENTER.**

Opción 2: Elegir uno de los tableros predeterminados

Al elegir uno de los tableros predeterminados se cargará un tablero con valores preestablecidos, de acuerdo al nivel de dificultad elegido.

La forma de cargar números sigue siendo a través de la botonera de la parte superior o utilizando el teclado.

Existen dos funcionalidades en el juego:

1) **Comprobar:** Informa si a partir de la disposición actual de números (sin quitar ningún número) es posible o no resolver el Sudoku.
2) **Resolver:** En caso de ser posible, resuelve el Sudoku a partir de la disposición actual de números.

Ejemplo Comprobar y Resolver.

Validación de los valores ingresados

Como se mencionó anteriormente el juego tiene una restricción y es que no debe repetirse ninguna cifra en una misma fila, columna o subcuadrícula (región) . Además, los números ingresados no pueden ser menores a 1 ni mayores a 9. Si una de estas situaciones ocurre, no se permitirá insertar el valor en la casilla deseada y se muestra el mensaje correspondiente.

Si el número ingresado no viola ninguna de las restricciones mencionadas, entonces queda guardado en la casilla elegida, y pasa a tener color rojo, a diferencia de los números predefinidos del tablero, que son de color negro.

Estos escenarios se ven reflejados en las siguientes imágenes:

¡Victoria!

Finalmente, cuando se logra resolver el Sudoku: