Proyecto 1: Sudoku

Lógica para ciencias de la computación

Profesores: Falappa, Marcelo y Gómez Lucero, Mauro.

Primer Cuatrimestre del 2017

Dímatz, Juan y Jouglard, Juan Francisco

Tabla de contenidos

Introducción

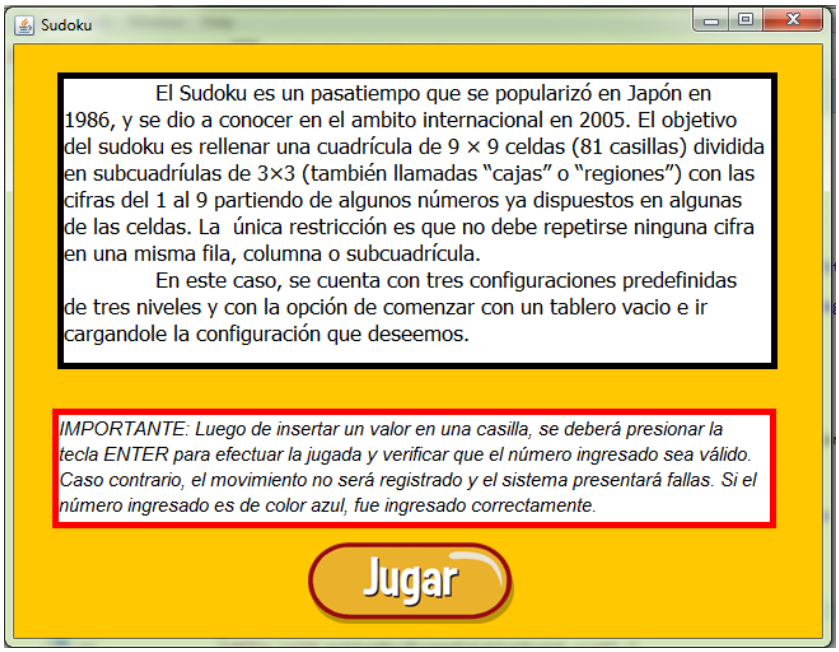
El Sudoku es un pasatiempo que se popularizó en Japón en 1986, y se dio a conocer en el ámbito internacional en 2005. El objetivo del sudoku es rellenar una cuadrícula de 9 × 9 celdas (81 casillas) dividida en subcuadrıculas de 3×3 (también llamadas “cajas” o “regiones”) con las cifras del 1 al 9 partiendo 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.

Hemos desarrollado un sistema que permite jugar como un usuario normal o resolverlo automáticamente. Además, se podrá comprobar si hay una solución posible con los números propuestos a medida que lo vamos completando.

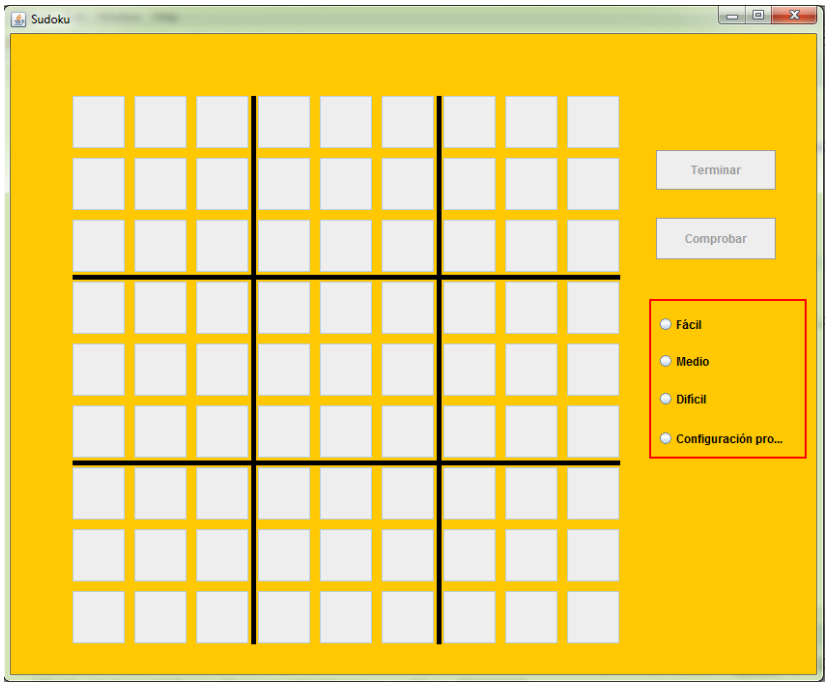
Se dispone de tres configuraciones predeterminadas para jugar, o se puede optar por empezar con un tablero totalmente vacío, y cargar una configuración preferida.

Interfaz gráfica y funcionalidades de la aplicación

Al comienzo de la ejecución del programa nos encontraremos con una pantalla que nos brindara la introducción antes dada y nos indicará las reglas del juego.

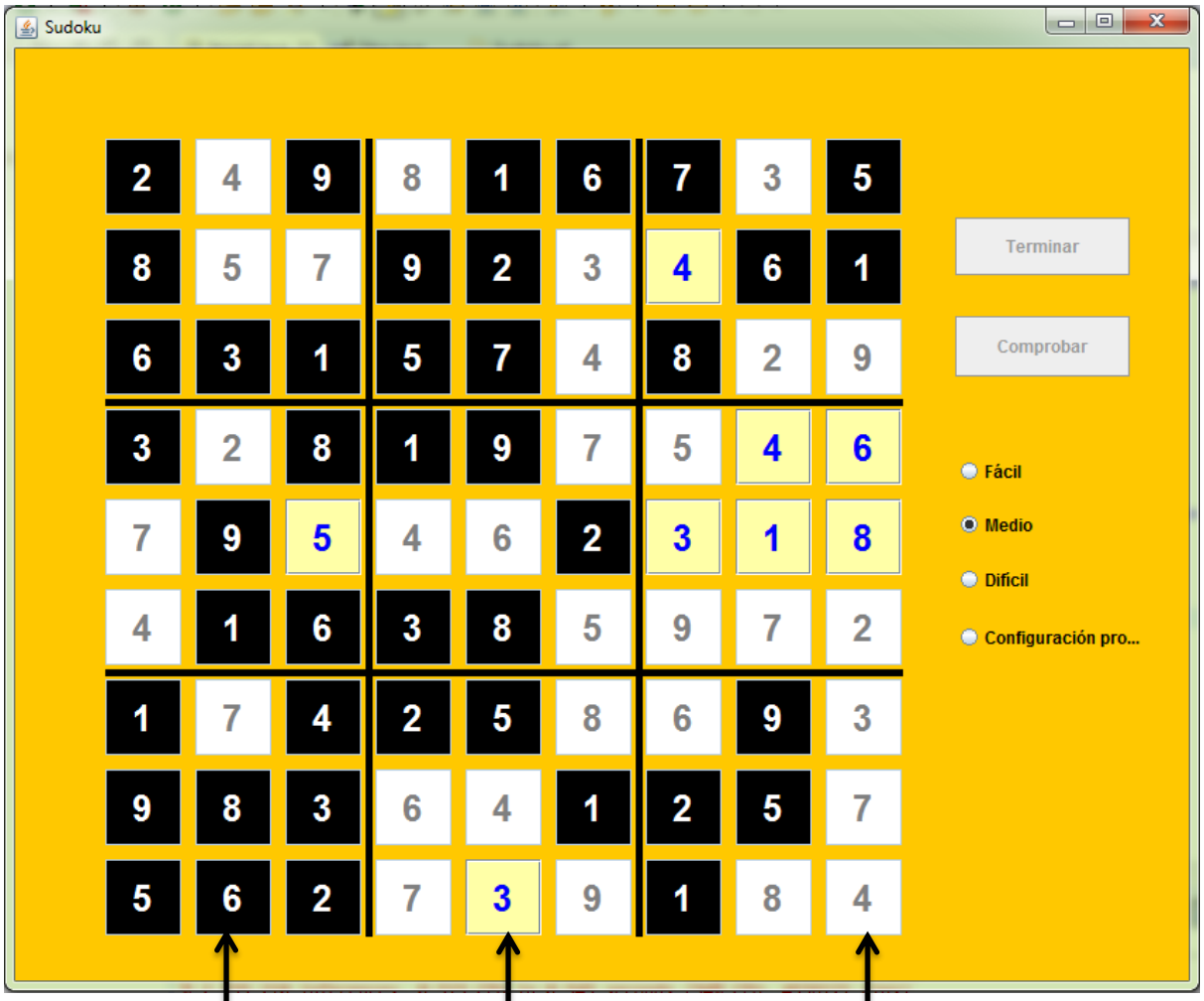


Luego de presionar el botón "Jugar", veremos una pantalla que representa la cuadrícula tradicional del Sudoku y dos botones "Resolver" y "Comprobar" (deshabilitados al comienzo), que nos permitirán realizar esas tareas. También habrá cuatro botones de opción que nos permitirán, elegir una de tres configuraciones con cifras dispuestas de antemano o cargar nuestra propia distribución de números. Inicialmente debemos elegir una de ellas.



Si decidimos jugar por nuestra cuenta, podremos completar las casillas vacías, ingresando un número y luego presionando “Enter”. Si el número no respeta las reglas del juego, es decir, si no está en el rango del 1 al 9, o está repetido en la fila, columna o bloque, se indicará mediante un mensaje por pantalla que hay un error. También podremos eliminar los números insertados.

En cualquier instancia del juego se puede hacer uso de las funciones de resolver o comprobar la disposición actual.



Numero insertado por el sistema cuando se le pidió resolución.

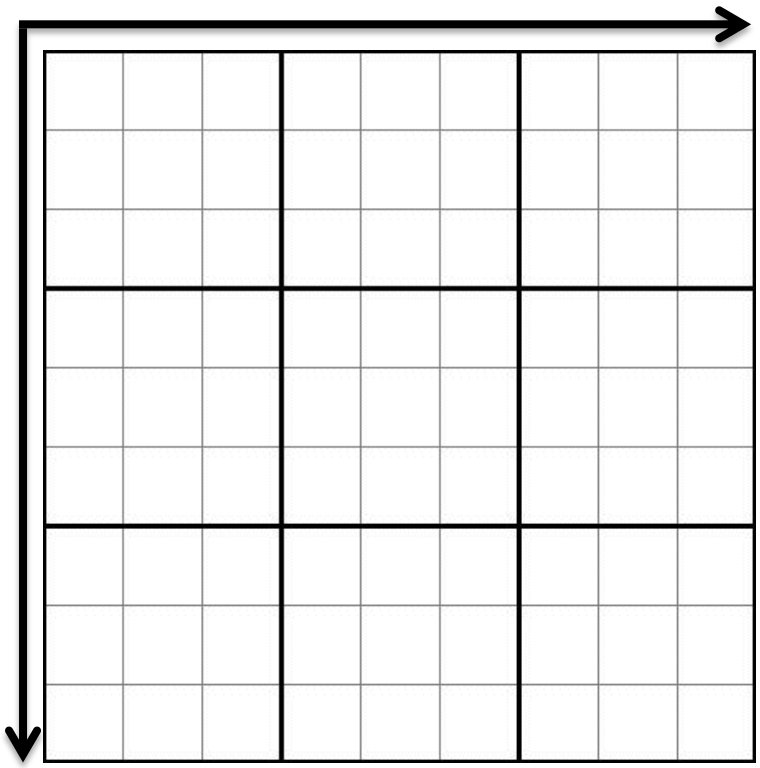
Numero insertado por el usuario.

Número de la configuración inicial.

Implementación del programa en Prolog

Decidimos usar una estructura de puntos, de la forma **punto(X,Y,N)**, para representar el tablero. En donde X es la columna, Y la fila y N el valor contenido. De esta forma, se simplifica la verificación de la validez de una movida, ya que, los números tienen coordenadas asociadas a ellos y podemos chequear si el número ingresado pertenece a la fila o columna, haciendo la consulta **punto(_,Y,N)** y **punto(X,_,N)** respectivamente. Por otra parte, mediante una fórmula matemática y el uso del predicado predefinido ***findall/3***, podemos obtener todas las posiciones de la caja en la que se quiere insertar y chequear las mismas.

El método para resolver, es recorrer las casillas de izquierda a derecha y de arriba para abajo, e ir poniendo números válidos en cada una de ellas. Si se llega a una casilla en la que ningún número puede ser ingresado, cambia las inserciones anteriores. Esto se realiza hasta que se llega a la última casilla (9,9) y se ingresa un número en ella. Si no encuentra ninguna solución valida, retorna el valor falso.



Mediante el uso del predicado **movimiento_valido(X,Y,N)**, podemos realizar dichas inserciones, llamándolo sin instanciar la variable N. De esta forma, tomara un número del 1 al 9 y lo insertará, verificando previamente que sea posible. Los números se asocian al tablero mediante el predicado **assert/1**, ya que, la estructura **punto/3**, es dinámica.

La forma de representar el conjunto de números es mediante hechos, que permitan seguir otro camino si el actual falla (**num(X)**, donde X es un número del 1 al 9).

Cada vez que no encuentra un número posible para una casilla, debe hacer back tracking y quitar los números que ingreso, para probar si hay solución por otro "camino". Por eso, no usamos el predicado predefinido asserta/1, sino uno definido por nosotros de la forma,

mi_asserta(punto(X,Y,N)):-asserta(punto(X,Y,N)).
mi_asserta(punto(X,Y,N)):-retract(punto(X,Y,N)),fail.

Así, cuando vuelve a buscar otro camino de ejecución, borra el número previamente ingresado y falla (sensible al backtracking), ya que nos interesa que ponga otro número en esa posición.

Conexión Java/Prolog

La librería JPL nos permite consultar predicados de Prolog desde Java, de esta forma, podemos conectar a la interfaz gráfica, con el tablero y las funcionalidades lógicas usadas para resolver el juego. Al momento de realizar una movida desde java (insertar un valor en la casilla), se llama al predicado **movimiento_valido/3** con las coordenadas y el número, para insertarlo en el tablero si es posible, el mismo se encargará de hacer todas las verificaciones necesarias.
Si el usuario solicita resolver o comprobar la configuración, se llama al predicado **solucion/0**. Si solo se pide comprobar, luego, se quitan los puntos insertados para la resolución.

Luego, para rellenar las casillas en pantalla, consultamos los hechos **punto(X,Y,N)**, con las coordenadas que queremos obtener, para luego mostrar el número N.
Idealmente, si el sudoku fue resuelto con éxito, al finalizar, la base de datos constará de 81 hechos **punto/3.**

Desde Java, no se ejecuta ningún tipo de procedimiento lógico para resolver el juego, solo se brinda una interfaz amigable que muestra la información brindada por el programa Prolog.

Estrategia que no fue implementada por dificultades

Tratamos de restringir los números elegibles a la hora de resolver, para que no pruebe con números que ya han sido ingresados y falle inminentemente. La forma que decidimos hacerlo fue, crear un nuevo predicado número/1, definido de la siguiente manera:

numero(X):-num(X),habilitado(X).

Donde el predicado **num/1** retornara un numero en X, y luego habrá que verificar que el mismo esté habilitado. Cuando se inserta un numero se deshabilita, para que no pueda ser usado en futuras pruebas. Luego, al cambiar de fila, todos los números son habilitados nuevamente.

Utilizamos el predicado habilitado (actúa como un "switch"), ya que, si hacemos assert y retract de los hechos **num/1**, cambiará el orden de los mismos, lo que generará fallas al momento de hacer backtracking y probar insertando diferentes números.

Esta estrategia, llevo a que la resolución de un Sudoku, cicle infinitamente, por lo que desistimos en usarla. Entendemos que esto se puede deber al excesivo uso de los predicados **assert/1** y **retract/1**.

Luego, intentamos implementarla con una lista en la que se guardaban los números insertados, pero esto resultaba en una resolución más ineficiente, por el contrario de su objetivo inicial.

Código del programa Prolog

```
%punto(X,Y,Numero)
:-dynamic punto/3.

%Hechos que representan el dominio de numeros con los que se trabajará.
num(1).
num(2).
num(3).
num(4).
num(5).
num(6).
num(7).
num(8).
num(9).

%Recibe unas coordenadas del tablero e inserta el numero N, con previa
verificacion en fila, columna y caja.
movimiento_valido(X,Y,N):-num(N),not(punto(_,Y,N)),not(punto(X,_,N)),Z is
ceiling(X/3),W is ceiling(Y/3),not(caja_con(Z,W,N)),mi_asserta(punto(X,Y,N)).

%Encuentro todas las posiciones de la caja y las recorro para encontrar N.
caja_con(Z,W,N):-posiciones(Z,W,L),recorrer(L,N).

posiciones(Z,W,Pos):-findall([X,Y],(num(X),num(Y),X=<Z*3,X>=(Z-
1)*3+1,Y=<W*3,Y>=(W-1)*3+1),Pos).

recorrer([[X,Y]|Zs],N):-punto(X,Y,N);recorrer(Zs,N).

%Asserta propio, sensible al backtracking.
mi_asserta(punto(X,Y,Z)):-asserta(punto(X,Y,Z)).
mi_asserta(punto(X,Y,Z)):-retract(punto(X,Y,Z)),fail.

%Predicado que resolvera el sudoku, recorriendolo de izquierda a derecha y de
arriba hacia abajo.
resolver(10,9).
resolver(10,Y):-Z is Y+1,resolver(1,Z).
resolver(X,Y):-not(punto(X,Y,_)),movimiento_valido(X,Y,_),Z is X+1,resolver(Z,Y).
resolver(X,Y):-punto(X,Y,_),Z is X+1,resolver(Z,Y).

solucion():-hay_posibilidad(1,1),resolver(1,1).

%Chequea que con la configuracion actual, en todas las casillas hay un numero
posible a insertar.
hay_posibilidad(10,9).
hay_posibilidad(10,Y):-Z is Y+1,hay_posibilidad(1,Z).
hay_posibilidad(X,Y):-
X<10,Y<10,not(punto(X,Y,_)),num(N),not(punto(_,Y,N)),not(punto(X,_,N)),Z is
ceiling(X/3),W is ceiling(Y/3),not(caja_con(Z,W,N)),Q is
X+1,!,hay_posibilidad(Q,Y).

hay_posibilidad(X,Y):-X<10,Y<10,punto(X,Y,_),Z is X+1,!,hay_posibilidad(Z,Y).
```