PROYECTO N°1

LOGICA PARA CIENCIAS DE LA COMPUTACION

SUDOKU

(en Prolog y Java)

Atuores:

AGRA, Federico G.

LOZA, Carlos I.

SUDOKU

QUE ES SUDOKU?

Sudoku (en japonés: 数独, sūdoku) es un juego matemático que se publicó por primera vez a finales de la década de 1970 y se popularizó en Japón en 1986, dándose a conocer en el ámbito internacional en 2005 cuando numerosos periódicos empezaron a publicarlo en su sección de pasatiempos. El objetivo del sudoku 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.[1]

COMO SE JUEGA?

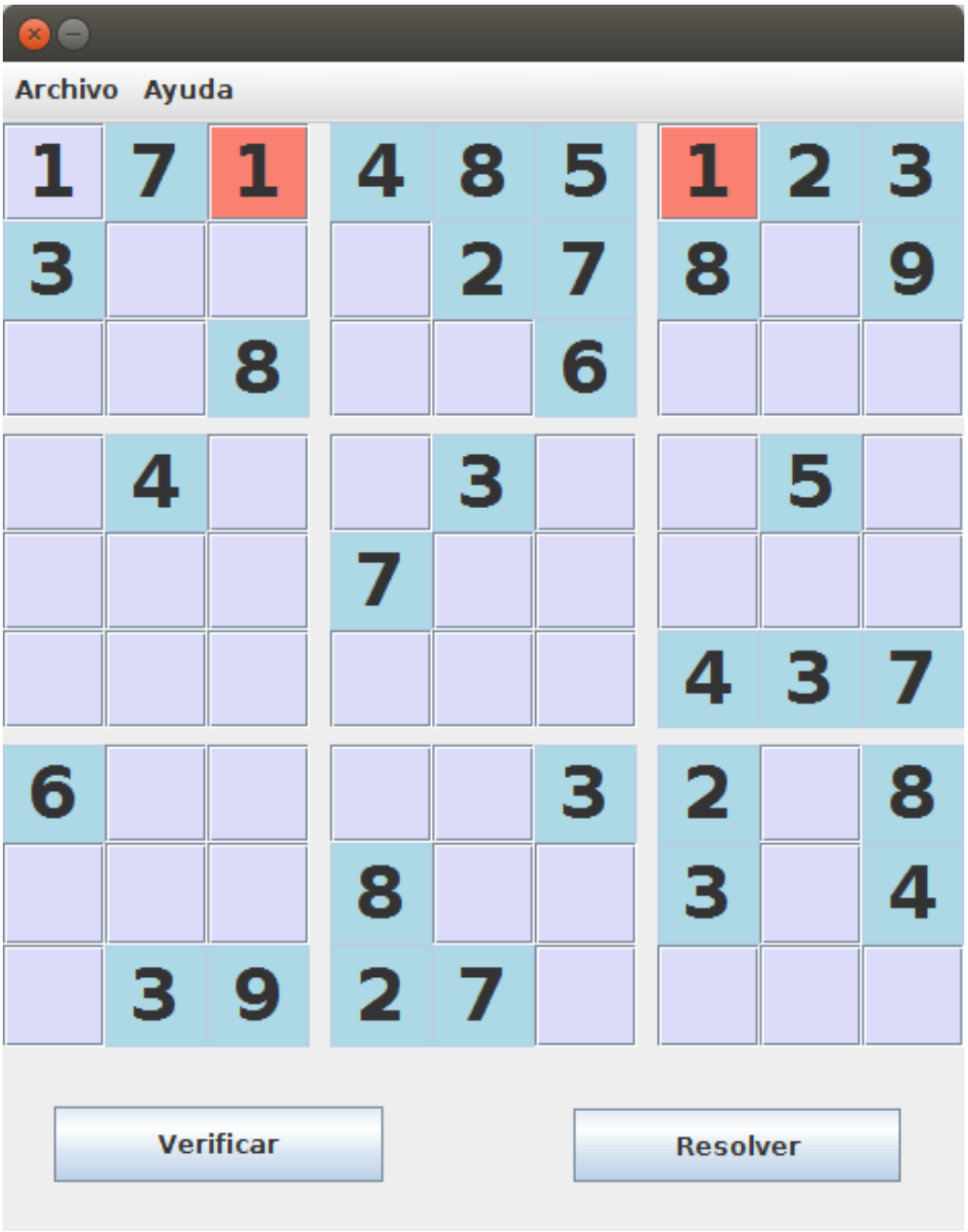
Para jugar sudoku se sebe completar el tablero (subdividido en nueve cuadrados) de 81 casillas (dispuestas en mueve filas verticales y horizontales) llenando los casilleros vacíos con los números de 1 al 9, de modo tal que no se repita ninguna cifra en ninguna fila ni columna, ni tampoco en cada cuadrado. A simple vista parece un juego complicado, pero es un juego un tanto sencillo, más de lo que parece.

[1] http://es.wikipedia.org/wiki/Sudoku

EL PROGRAMA

INTERFAZ GRAFICA

El juego presenta una interfaz gráfica un tanto amigable y sencilla tanto en uso como en comprensión de funcionalidad. Consta de tablero donde eventualmente se desarrollará el juego, en el cual se aprecian las casillas en sus respectivas subdivisiones. Además se puede apreciar una barra de herramientas con menú de opciones y dos botones especiales referentes a la funcionalidad del juego.



FUNCIONES

Inicialmente el juego presenta un tablero sin resolver donde podremos comenzar a jugar. Para ello deberemos ir llenando cada casillero sin olvidar la regla principal (no se permiten repeticiones en filas, columnas y subdivisiones), en caso de ingresar erróneamente algún digito que cree conflicto en su columna, fila o subdivisión el programa nos alertara marcando del error cometido.

Sumando a lo presentado el juego posee dos botones de gran utilidad: "Verificar" y "Resolver". Una vez llenado tablero por completo, el botón "Verificar" comprueba que todos los casilleros hayan sido completados correctamente respetando la regla principal del juego, advirtiendo de errores o indicándonos una buena jugada final. Aun así podemos darnos dotes de perezosos y dejar que el juego resuelva el tablero por su cuenta, aquí es donde entra en acción el botón "Resolver" completando por nosotros el tablero de forma correcta.

Concluyendo con la funcionalidad disponible podremos apreciar una barra de tareas con menú desplegable, donde encontraremos opciones para carga de juegos manualmente, como así también el "acerca de" con información respecto a los desarrolladores del programa.

IMPLEMENTACION

CONDICIONES DE IMPLMENTACION

Como condición para para este proyecto, se encontraba la realización de la lógica del juego meramente en lenguaje Prolog, y la implementación de la interfaz gráfica en Java. Si bien la implementación de la lógica es un tanto sencilla (en comparación con la gráfica) no la hace menos importante, y aunque la gráfica se vea humilde costo tiempo y dedicación llegar a su meta.

LOGICA EN PROLOG

La lógica en Prolog como ya lo anticipamos es sencilla (no así la solución, que fue difícil dar con ella). La lógica consta con tres predicados básicos que componen toda la lógica del juego: comprobar, resolver y validar (existen más predicados para resolución de los anteriores).

Comencemos por el predicado más básico, el predicado validar, el cual recibe una lista y verifica que ninguno de sus componentes se repita, ¿fácil verdad?, además este métodos hace uso del predicado dif de aridad 2 que posee una propiedad la cual restringe el valor que podrán tomar las metas evitando en un futuro sean iguales, la utilidad de esta propiedad se argumenta más adelante

Agregando un poco de complejidad nos encontramos con el método comprobar, que recibe un tablero de 9x9 (representado en una lista de 81 componentes) y verifica que todos los casilleros no tengan conflicto en su columna, fila o subdivisión; estos se logra realizando tres grupos de comprobaciones, verificando de a 9 componentes: 9 componentes de cada fila, 9 de cada columna y 9 de cada subdivisión, dicha comprobación se realiza con el predicado "validar" mencionado anteriormente. Llegado el punto en que ningún caso halla conflicto el predicado devolverá verdadero, caso contrario devolverá falso.

Concluyendo con la lógica del juego, nos encontramos con el predicado resolver, que tiene un funcionamiento similar a Verificar, en este caso se recibe un tablero a medias y usando el predicado validar se recorrerán todas la componentes y restringiendo aquellas que sean vacías (con el predicado dif) así para cada fila, columna y subdivisión; luego el predicado labelear ira asignando aquellos valores que sean posibles hasta completar el tablero, notar que cada casillero no es completamente restringido y que probablemente posea más de un valor posible, si la combinación en todo el tablero es válida y libre de conflicto el predicado retorna una lista con los valores correctos, si la combinación es errada el procesos de asignación se repite y la comprobación se realiza nuevamente hasta dar con una secuencia correcta. En caso de que inicialmente se disponga una configuración imposible de resolver el predicado nos alertará de tal situación retornando falso.

INTERACCION JAVA-PROLOG

La interacción entre la interfaz en java y la lógica en Prolog se logra gracias a librería jpl.jar destinada a tal fin, permitiendo crear objetos de tipo "consulta" a la cual le podremos pedir información tales como si el predicado resultó verdadero o falso o podremos consultar por un conjunto de posibles soluciones para una variable no definida.

Desde Prolog todo funciona sin cambios, tal cual lo hace en swi-prolog. Desde la interfaz en java se maneja mediante String, las consultas son enviadas como tal y las respuestas vuelven del mismo modo, por lo que para manejar las soluciones hay que hacer manejos de caracteres, en especial para desmenuzar la solución del tablero resuelto y del mismo modo para armar el String que conformar el predicado

RUTINA DE PROLOG

```
/**
SudokuFinal(V2).pl

Logica en prolog para jugar una partida de sudoku.
Esta version es una version mas refinada pero no mas eficiente ya que presinde del uso
de la libreria "clpfd" que optimizaba la asignacion de los valores en el trablero del
juego. como se anunció esta version no es mas eficiente que la anterior pero
si es mas facil de comprender el funcionamiento de la logica. En resumidas palabras
los predicados en prolog de este programa verifican que en cada fila, columna y
subdivision sean las cifras del 1 al 9 sin repetirse,esto se hará tanto hace para la
verificacion de un tablero ya completo, como asi tambien para la comprobacion de un
tablero a completar, con la salverdad de que en un tablero a completar se iran
asignando los valores y luego se verificaran que esto sean correctos, de lo contrario
se asignan valores nuevos y nuevamente se verifican, esto sucesivamente hasta dar con
la solucion correcta, en caso de existir se retorna la solucion, o se acusa de ello en
caso contrario.
puede obtener el codigo fuente de la version anterior en:
https://code.google.com/p/agra-loza-project/source/

autores:
       AGRA, Federico G.
       LOZA, Carlos I.

03 de junio de 2014
*/

/**
metodo que comprueba si un tablero esta correctamente armado verificando en cada fila
columna y subdivision no se repitan las cifras del 1 al 9

 input:
       @Salid: una lista de 81 componentes, que eventualmente
       representa la matriz de 9x9 de un tablero de sudoku, esta
       lista ingresa previamente cargada
 output:
       True si la lista es correcta
       False caso contrario
*/
comprobar(Salid):-
       /*Establezco formato*/
       Salid =[A1,A2,A3,  A4,A5,A6,  A7,A8,A9,
                     B1,B2,B3,  B4,B5,B6,  B7,B8,B9,
```

```
                    C1,C2,C3,  C4,C5,C6,  C7,C8,C9,

                    D1,D2,D3,  D4,D5,D6,  D7,D8,D9,
                    E1,E2,E3,  E4,E5,E6,  E7,E8,E9,
                    F1,F2,F3,  F4,F5,F6,  F7,F8,F9,

                    G1,G2,G3,  G4,G5,G6,  G7,G8,G9,
                    H1,H2,H3,  H4,H5,H6,  H7,H8,H9,
                    I1,I2,I3,  I4,I5,I6,  I7,I8,I9],

      /*comprobar las filas*/
      validar([A1,A2,A3,A4,A5,A6,A7,A8,A9]),
      validar([B1,B2,B3,B4,B5,B6,B7,B8,B9]),
      validar([C1,C2,C3,C4,C5,C6,C7,C8,C9]),
      validar([D1,D2,D3,D4,D5,D6,D7,D8,D9]),
      validar([E1,E2,E3,E4,E5,E6,E7,E8,E9]),
      validar([F1,F2,F3,F4,F5,F6,F7,F8,F9]),
      validar([G1,G2,G3,G4,G5,G6,G7,G8,G9]),
      validar([H1,H2,H3,H4,H5,H6,H7,H8,H9]),
      validar([I1,I2,I3,I4,I5,I6,I7,I8,I9]),

      /*comprobar las columnas*/
      validar([A1,B1,C1,D1,E1,F1,G1,H1,I1]),
      validar([A2,B2,C2,D2,E2,F2,G2,H2,I2]),
      validar([A3,B3,C3,D3,E3,F3,G3,H3,I3]),
      validar([A4,B4,C4,D4,E4,F4,G4,H4,I4]),
      validar([A5,B5,C5,D5,E5,F5,G5,H5,I5]),
      validar([A6,B6,C6,D6,E6,F6,G6,H6,I6]),
      validar([A7,B7,C7,D7,E7,F7,G7,H7,I7]),
      validar([A8,B8,C8,D8,E8,F8,G8,H8,I8]),
      validar([A9,B9,C9,D9,E9,F9,G9,H9,I9]),

      /*comprobar las cuadriculas*/
      validar([A1,A2,A3,B1,B2,B3,C1,C2,C3]),
      validar([A4,A5,A6,B4,B5,B6,C4,C5,C6]),
      validar([A7,A8,A9,B7,B8,B9,C7,C8,C9]),
      validar([D1,D2,D3,E1,E2,E3,F1,F2,F3]),
      validar([D4,D5,D6,E4,E5,E6,F4,F5,F6]),
      validar([D7,D8,D9,E7,E8,E9,F7,F8,F9]),
      validar([G1,G2,G3,H1,H2,H3,I1,I2,I3]),
      validar([G4,G5,G6,H4,H5,H6,I4,I5,I6]),
      validar([G7,G8,G9,H7,H8,H9,I7,I8,I9]).

/**
predicado que resuelve un tablero de sudoku a partir de una par de cifras ya cargadas,
el metoco comprobar verifica que las cifras no posean conflito y llamada al metodo
lavelear/1 para que asigne valores a la lista, el proceso funciona a fuerza bruta,
probando combinaciones hasta dar con una que satisface la condicion de comprobar/1.

input:
      @Tablero: lista de 81 coponentes con algunas de sus componentes completas el
      cual representa la matriz del sudoku a resolver
output:
      @salida: una lista de 81 componentes con, en caso de existir, la solucion
      al sudoku ingrasado
      False: en caso de que el sudoku propuesto no posea solucion alguna
*/
resolver(Tablero, Salida):-

      Salida=Tablero,

      comprobar(Salida),
```

```
	labelear(Salida)
	.

/**
 predicado que verifica una lista corroborando que sus componentes sean todos
distintos, si bien este metodo podria no existir y usarse directaente el método
todosDiferentes/1, se conserva para no modificar la interfaz grafica en java que ya se
verifico su correcto funcionamiento.

imput:
	@L: lista a la cual se debera corroborar que sus componentes sean distintas
output:
	True: si la lista posee todos sus componentes sos distitas
	False: caso contrario
*/
validar(L) :-
	todosDiferentes(L).

/* */
labelear([]).
labelear([L|Ls]):-
	member(L,[1,2,3,4,5,6,7,8,9]),
	labelear(Ls).

/**
predicado que verifica si dos metas-variables son distintas, este predicado hace uso
del predicado dif/2 de la libreria dif que se autocarga con prolog, estre predicado
roporciona una restricción que indica que A y B son términos diferentes. Si A y B no
pueden unificar, dif/2 tiene éxito determinista. Si A y B son idénticos falla
inmediatamente, y, por último, si A y B se pueden unificar, las metas se retrasan,
impidiendo a A y B a ser iguales.
Mas info: http://www.swi-prolog.org/pldoc/man?predicate=dif/2

imput:
	@X, @y: variables con las cuales se comprobara la igualdad y posterior
restriccion
output:
	False: si ambas metas son iguales
	True: caso contrario
*/
diferentes(X,Y):-
	dif(X,Y).

/**
Predicado recursivo que verifica que todas las componentes de una lista sean distintos
en el caso base una lista es vacia, lo cual marca como verdadero que cumple la
condicion, en el caso recursivo verifico que el primer elemento sea distitnto al resto
de la lista y luego verifico que el resto de la lista posea todas sus componentes
distintas

input:
	@[] lista vacia
	@[L|Ls] lista donde L es la cabeza de la lista y Ls es el resto de la lista
output:
	True: si las componestes son todas distintas
	False: caso contrario
*/
todosDiferentes([]).
todosDiferentes([L|Ls]):-
	diferentePrimero(L,Ls),
	todosDiferentes(Ls).

/**
```

```
Predicado recursivo que corrobora que un elemento dado no se encuentre en una list
tambien dada, en el caso base ferefico que un elemto y una lista vacias, lo cual marca
como verdadero cumpliendo la condicion, y en el caso recursivo se verifica la
desigualdad entre el elemento X dado y la cabeza Y de la lista, luego se verifica que
el elemento dado sea distinto al resto de la lista dada

input:
	@X: elemento dado que no deve existir en la lista
	@[L|Ls]: lista donde se realizara de prueva, L cabeza de la lista, Ls resto de
la lista
	@_: elemento sin considerar contenido
	@[]: lista vacia
*/
diferentePrimero(_,[]).
diferentePrimero(X,[Y|Ls]):-
	diferentes(X,Y),
	diferentePrimero(X,Ls).
```