Proyecto N°1

Lógica para Ciencias de la Computación

FlickColor

(Prolog en Web**)**

Autores:

Christian Rubén Bancalá

Carlos Ignacio Loza

FlickColor

Que es FlickColor?

FlickColor es un interesante juego de acertijos para todos aquellos a quienes les gustan los juegos con varios colores. El objetivo es limpiar la Grilla de sus elementos coloridos, eliminando la diversidad de estos colores logrando pintar todo de un único color.

El Juego

Consta de una grilla de 14 × 14, donde cada celda está pintada de uno de 6 posibles colores, y 6 botones, uno de cada color, como se muestra en la figura. Presionar un botón de color C causa que la celda superior izquierda de la grilla, pintada de color C', al igual que todas las del mismo color C' adyacentes a esta, y las de color C' adyacentes a éstas últimas, y así siguiendo, se pinten de color C. Notar que si C=C' entonces no se produce cambio alguno, por lo tanto se asume que presionar el botón del mismo color que la celda superior izquierda no constituye una jugada válida.

Objetivo del Juego

El objetivo del juego es lograr pintar todas las celdas de un mismo color, presionando la botonera de colores la menor cantidad de veces posible. No te preocupes por la complejidad!! Incorporamos ayudas para facilitarte la experiencia.

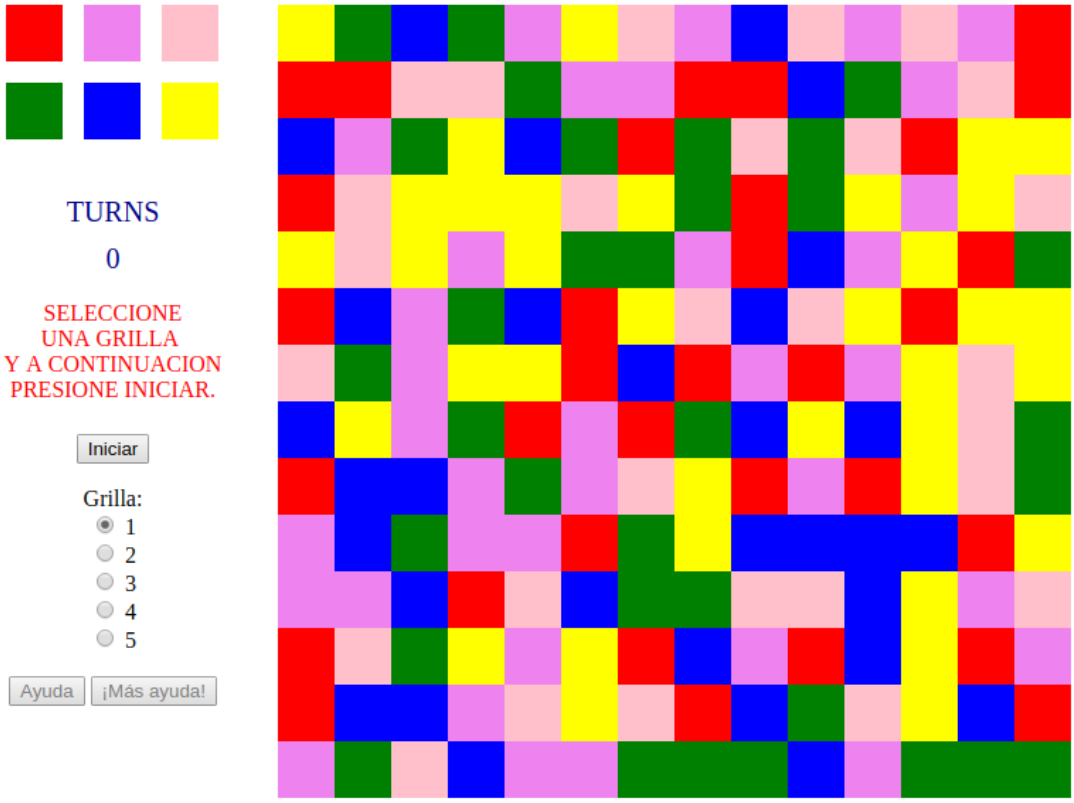
Observación antes de continuar

Notación: usa remos adyacenteC* para referirnos a la clausura transitiva de la relación de adyacencia entre dos celdas del mismo color. Es decir, diremos que una celda X es adyacenteC* a una celda Y si ambas son del mismo color y además X es o bien adyacente a Y , o existe una celda Z del mismo color que X e Y tal que X es adyacente a Z y recursivamente Z es adyacenteC* a Y.

La Pagina Web

Vista

El juego presenta una pagina web un tanto amigable y sencilla tanto en uso como en comprensión de funcionalidad. Consta de tablero o grilla 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 a izquierda con menú de opciones para el desarrollo del juego y dos botones especiales al final referentes a las ayudas.



Funciones

Inicialmente el juego presenta un tablero sin resolver, entre varios disponibles para elegir,donde podremos comenzar a jugar. Para ello deberemos seleccionar la grilla que queremos jugar y luego iniciar.

El juego se desarrolla al ir seleccionando alguno de los seis colores del panel izquierdo sin olvidar la que desde la esquina izquierda superior se aplicara la regla adyacenteC* para el nuevo color seleccionado (para una idea global y de simple comprensión el nuevo color se esparcirá hacia todas las casillas del mismo color que se toquen entre si partiendo desde la esquina superior izq).

Por ultimo al final contamos con dos ayudas para facilitar el juego en cualquier momento, la primera “ayuda” nos indicará el beneficio de presionar cada color, es decir, cuantas casillas se van a incorporar a los adyacenteC* de la esquina superior izq, la segunda “mas ayuda!” es un función similar con la diferencia que nos mostrara el beneficio de jugar dos veces comenzando por el color en el panel de selección y luego la mejor siguiente jugada

Implementación

Condiciones de implementación

Como condición para este proyecto, se encontraba la realización de la lógica del juego meramenteen lenguaje Prolog, y la implementación de la interfaz gráfica en por medio de una pagina web. Si bien la implementación de la lógica puede parecer, a simple vista, un tanto extensa (en comparación con la gráfica), no es una logica dificil de comprender, no así la hace menos importante, y aunque la gráfica se vea humilde costo tiempo y dedicación llegar a su meta.

Lógica en PROLOG

La logica en prolog como ya se antisipó es un poco extensa pero no es dificil de entender, todo el flujo de datos hacia la grafica es a travez de 5 predicados, el principal es flick/3 que realiza la jugada según el color seleccionado, grid/2 que permite elegir la grilla a jugar, los dos predicados ayudaBasica/2 y ayudaAdicional/2 que proporcionan la ayuda al jugador y por ultimo el predicado verificarVictoria/2. Cabe mencionar que si bien los predicados antes mensionados son la "interfaz" el calculo de fondo se realiza por medio de dos predicados importantes, a nombrar, pintar/7 y pintarContorno/7

Comenzando por el predicado mas facil, grid/2, este recibe un numero entero y retorna una grilla con una distribucion de colores correspondiente a esa configuracion pre-establecida.

Luego sigue el predicado el predicado flick/3 que, repitiendo lo mensionado, realiza la jugada según el color seleccionado. Esta funcionalidad se realiza por medio del predicado pintar/7 al cual de forma recursiva va cambiando de color las celdas adyacentesC* a la celda superior izquierda, y al finalizar retorna la grilla resultante,es decir, la grilla modificada, la cual diremos que se le hizo flick con el color ingresado. A su vez este movimiento de hacer flick es sencillo de entender, si pensamos en la propiedad adyacenteC* donde dos celdas X e Y son directamente adyacentes o bien existe una celda Z donde esta es directamente ayacente a X y cumple la propiedad adyacenteC* con Y, con esta idea nos trasladamos a pintar/2, que recive una coordenada en la grilla ubicando la celda X, realizando tres comprobaciones, primero corroboramos logicamente que la coordenada no apunta fuera de la grilla, luego corroboro que no sea un color distinto a la celda superior izquierda (dicho color viaja a travez de uno de los paramatros para ganar en eficiencia); por ultimo si esta celda es del color de la celda sup izq le asigno el color nuevo y llamo recursivamente con las 4 celdas adyacentes (buscando celdas que cumplen las veces de celda Z) a travez del predicado pintarContorno/7 que apricará pintar/7 a estas celdas, nuevamente se realizaran las anteriores comprobaciones y proceso continua hasta no encontrar celdas adyacentes que pintar, esto es que no cumplen con adyacentesC*. En este proceso se utilizan predicados auxiliares getColorEnPos/4, getColorEnLista/3, reemplazarEnLista/3, cambiarColorEnPosicion/5, largo/2 que cumplen tareas basicas e importantes y que no detallaremos ya que el nombre de cada predicado es bastante significativo. Por ultimo detalle a mensionar de este predicado es que cuenta con un parámetro que retorna la cantidad de celdas pintadas en el proceso de hacer flick a la grilla, este parámetro no hace a la funcionalidad de la resolución de las jugadas pero sirve para las ayudas y se detalla su uso mas adelante.

Ademas, contamos con los dos predicados de ayuda al jugador, a nombrar, ayudaBasica/2 y ayudaAdicional/2. Comenzando por el primero y mas sencillo de entender, este predicado es una cascara de ayudaBasica2/2, este ultimo recibe una lista lista de los colores posibles y retorna una lista de beneficio para uno de estos, recursivamente, llama al predicado cantIncorpora/3 que realizara dos veces pintar/2 guardando en cada llamada el parámetro cantidad antes mencionado, la primera vez "pinta" del color que recibe de la lista y la segunda vez del color original, luego devuelve la diferencia de ambas cantidades que denota el beneficio de pintar ese color, al finalizar la grilla resultante es descartada, por lo que no se aplican cambios en la grilla antes de pedir la ayuda; si se pide el beneficio de pintar el mismo color que la celda superior izquierda el beneficio lógicamente es ninguno, que matemáticamente es cero. El segundo predicado de ayuda, ayudaAdicional/2, devuleve el mejor beneficio de dos jugadas, esto no es mas que aplicar ayudaBasica/2 dos veces, en la primer llamada obtengo el beneficio de cada color en la primer

jugada, en las grillas resultantes aplico nuevamente ayudaBasica2 y solo me quedo con el mejor de estos beneficios para la segunda jugada, luego sumo ambos beneficios y obtengo para cada color el mejor beneficio de jugar dos jugadas consecutivas.

Terminando con la explicación de los predicados, concluimos con verificarVictoria/2 que recibe una grilla, la pinta de amarillo y la compara con una grilla completa de amarillo, si ambas grillas son iguales retorna 1 denotando la victoria, caso contrario 0.

Interacción PROLOG-Web

La interacción entre la interfaz (pagina web) y la lógica en Prolog se logra gracias a Pengine destinada a tal fin, permitiendo desde javaScript crear “consultas” a la cual le podremos pedir información a los predicados pudiendo consultar por posibles soluciones para una variable no definida.

Desde Prolog todo funciona sin cambios, tal cual lo hace en swi-prolog. Desde javaScript 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

```
:- module(proylcc,
       [
          grid/2,
          flick/3,
          ayudaBasica/2,
          ayudaAdicional/2,
          verificarVictoria/2
       ]).

grid(1, [
           [y,g,b,g,v,y,p,v,b,p,v,p,v,r],
           [r,r,p,p,g,v,v,r,r,b,g,v,p,r],
           [b,v,g,y,b,g,r,g,p,g,p,r,y,y],
           [r,p,y,y,y,p,y,g,r,g,y,v,y,p],
           [y,p,y,v,y,g,g,v,r,b,v,y,r,g],
           [r,b,v,g,b,r,y,p,b,p,y,r,y,y],
           [p,g,v,y,y,r,b,r,v,r,v,y,p,y],
           [b,y,v,g,r,v,r,g,b,y,b,y,p,g],
           [r,b,b,v,g,v,p,y,r,v,r,y,p,g],
           [v,b,g,v,v,r,g,y,b,b,b,b,r,y],
           [v,v,b,r,p,b,g,g,p,p,b,y,v,p],
           [r,p,g,y,v,y,r,b,v,r,b,y,r,v],
           [r,b,b,v,p,y,p,r,b,g,p,y,b,r],
           [v,g,p,b,v,v,g,g,g,b,v,g,g,g]
          ]).

grid(2, [
           [y,y,b,g,v,y,p,v,b,p,v,p,v,r],
           [y,y,p,p,g,v,v,r,r,b,g,v,p,r],
           [b,y,g,y,b,g,r,g,p,g,p,r,y,y],
           [r,y,y,y,y,p,y,g,r,g,y,v,y,p],
           [y,p,y,v,y,g,g,v,r,b,v,y,r,g],
           [r,b,v,g,b,r,y,p,b,p,y,r,y,y],
           [p,g,v,y,y,r,b,r,v,r,v,y,p,y],
           [b,y,v,g,r,v,r,g,b,y,b,y,p,g],
           [r,b,b,v,g,v,p,y,r,v,r,y,p,g],
           [v,b,g,v,v,r,g,y,b,b,b,b,r,y],
           [v,v,b,r,p,b,g,g,p,p,b,y,v,p],
           [r,p,g,y,v,y,r,b,v,r,b,y,r,v],
           [r,b,b,v,p,y,p,r,b,g,p,y,b,r],
           [v,g,p,b,v,v,g,g,g,b,v,g,g,g]
          ]).

grid(3, [
           [y,y,b,g,v,y,p,v,b,p,v,p,v,r],
           [y,y,p,p,g,v,v,r,r,b,g,v,p,r],
```

```
            [p,g,v,y,y,r,b,r,v,r,v,y,p,y],
            [b,y,v,g,r,v,r,g,b,y,b,y,p,g],
            [b,y,g,y,b,g,r,g,p,g,p,r,y,y],
            [r,b,b,v,g,v,p,y,r,v,r,y,p,g],
            [r,p,g,y,v,y,r,b,v,r,b,y,r,v],
            [r,b,b,v,p,y,p,r,b,g,p,y,b,r],
            [r,y,y,y,y,p,y,g,r,g,y,v,y,p],
            [y,p,y,v,y,g,g,v,r,b,v,y,r,g],
            [r,b,v,g,b,r,y,p,b,p,y,r,y,y],
            [v,b,g,v,v,r,g,y,b,b,b,b,r,y],
            [v,v,b,r,p,b,g,g,p,p,b,y,v,p],
            [v,g,p,b,v,v,g,g,g,b,v,g,g,g]
           ]).

grid(4, [
            [g,y,b,g,v,y,p,v,b,p,v,p,v,r],
            [r,g,p,p,g,v,v,r,r,b,g,v,p,r],
            [b,v,g,y,b,g,r,g,p,g,p,r,y,y],
            [r,p,y,g,y,p,y,g,r,g,y,v,y,p],
            [y,p,y,v,g,y,g,v,r,b,v,y,r,g],
            [r,b,v,g,b,g,y,p,b,p,y,r,y,y],
            [p,g,v,y,y,r,g,r,v,r,v,y,p,y],
            [b,y,v,g,r,v,r,g,b,y,b,y,p,g],
            [r,b,b,v,g,v,p,y,g,v,r,y,p,g],
            [v,b,g,v,v,r,g,y,b,g,b,b,r,y],
            [v,v,b,r,p,b,g,g,p,p,g,y,v,p],
            [r,p,g,y,v,y,r,b,v,r,b,g,r,v],
            [r,b,b,v,p,y,p,r,b,g,p,y,g,r],
            [v,g,p,b,v,v,g,g,g,b,v,g,r,g]
           ]).

grid(5, [
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y],
            [y,y,y,y,y,y,y,y,y,y,y,y,y,y]
           ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% flick(+Grid, +Color, -FGrid)
%
% FGrid es el resultado de hacer 'flick' de la grilla Grid con el color Color.
% En caso de que el color sea igual al color adyacente actual, se retorna la misma grilla.
flick(Grid,Color,Grid):-
	%este es una suerte de "caso base" para cuando ingresa el mismo color que la esquina sup izq
	Grid = [F|_],
	F = [Color|_].
flick(Grid,Color,FGrid):-
	Grid = [F|_],
	F = [X|_],
	pintar(X,Color,Grid,0,0,FGrid,_).
	%FGrid = [[Color|Xs]|Fs].

%metodo para pintar en un matriz de color "Color" las celda ingresada
%y sus adyacentes si estas eran el color "Ant"
pintar(Ant,Color,Grid,X,Y,Rta,CantPintado):-
	getColorEnPos(Grid,X,Y,PosCol),
	Ant=PosCol,
	cambiarColorEnPosicion(Color,Grid,X,Y,RtaA),
	pintarContorno(Ant,Color,RtaA,X,Y,Rta,CPTotal),
	%agrego este contador para la ayuda
```

```
	CantPintado is CPTotal+1.
pintar(Ant,_,Grid,X,Y,Grid,0):-
	%aca controlo que no sea "adyacenteC*", es decir, no es del mismo color
	getColorEnPos(Grid,X,Y,PosCol),
	Ant\=PosCol.
pintar(_,_,[G|Grid],X,Y,[G|Grid],0):-
	%aca controlo que no se me caiga de la grilla
	X<0;
	Y<0;
	largo([G|Grid],LF),    X>=LF;
	largo(G,LC), Y>=LC.

%metodo para pintar las celdas directamente adyacente a un celdas
%arriba, abajo, izquierda y derecha
pintarContorno(Ant,Color,Grid,X,Y,Rta,CantPintado):-
	Xmen is X-1,Ymen is Y-1,
	Xmas is X+1,Ymas is Y+1,
	pintar(Ant,Color,Grid,Xmen,Y,RtaA,CPA),
	pintar(Ant,Color,RtaA,Xmas,Y,RtaB,CPB),
	pintar(Ant,Color,RtaB,X,Ymas,RtaC,CPC),
	pintar(Ant,Color,RtaC,X,Ymen,Rta,CPD),
	%agrego este contador para la ayuda
	CantPintado is CPA+CPB+CPC+CPD.

%metodo recursivo para obtener el elemento en una posicion de la matriz
getColorEnPos([G|_],X,0,Rta):-
	getColorEnLista(G,X,Rta).
getColorEnPos([_|Grid],X,Y,Rta):-
	YY is Y-1,
	getColorEnPos(Grid,X,YY,Rta).

%metodo recursivo para obtener el elemento en una posicion de la lista
getColorEnLista([L|_],0,L).
getColorEnLista([_|Ls],X,Rta):-
	XX is X-1,
	getColorEnLista(Ls,XX,Rta).

%metodo recursivo para cambiar un elemento en un lista
%segun la posicion ingresada
reemplazarEnLista(Color,[_|Ls],0,[Color|Ls]).
reemplazarEnLista(Color,[L|Ls],X,[L|Rta]):-
	XX is X-1,
	reemplazarEnLista(Color,Ls,XX,Rta).

%metodo recursivo para cambiar un elemento en un matriz
%segun la coordenada ingresada
cambiarColorEnPosicion(Color,[G|Grid],X,0,[Rta|Grid]):-
	reemplazarEnLista(Color,G,X,Rta).
cambiarColorEnPosicion(Color,[G|Grid],X,Y,[G|Rta]):-
	YY is Y-1,
	cambiarColorEnPosicion(Color,Grid,X,YY,Rta).

%metodo para calcular el largo de una lista
largo([],0).
largo([_|Xs],Rta):- largo(Xs,Rtaa),
	Rta is Rtaa+1.

%hecho para retorna una lista de los colores posibles, sino los tengo que escribir
%a cada rato
colores(["r","v","p","g","b","y"]).

%metodo para calcular cuantas celdas se incorporan al cambiar a un cierto color
cantIncorpora(Grid,Color,0):-
	getColorEnPos(Grid,0,0,Color).
cantIncorpora(Grid,Color,Rta):-
	Grid = [F|_],
	F = [X|_],
	pintar(X,Color,Grid,0,0,FGrid,RtaA),
	pintar(Color,X,FGrid,0,0,_,RtaB),
	Rta is (RtaB-RtaA).

/*
```

```
Un metodo cascara que retorna una lista con cada uno de los adyacentes que se agregan a la
lista, haciendo uso del metodo cantIncorpora.
*/
ayudaBasica(Grid,Rta):-
       colores(Colores),
       ayudaBasica2(Grid,Colores,Rta).

ayudaBasica2(_,[],[]).
ayudaBasica2(Grid,[Color|Ls],[R|Rta]):-
       cantIncorpora(Grid,Color,R),
       ayudaBasica2(Grid,Ls,Rta).
/*
Este metodo retorna una ayuda aun mayor que la básica. Mediante un primer color, se averigua
cuantas celdas adyacentes nuevas pueden obtenerse en caso de apretar dicho primer boton y luego
cualquiera de los otros seis.
*/
ayudaAdicional(Grid,Rta):-
       colores(Colores),
       ayudaBasica(Grid,L),
       ayudaAdicional2(Grid,L,Colores,Rta).

ayudaAdicional2(_,[],[],[]).
ayudaAdicional2(Grid,[Cant | Rs],[Color|Ls],[R|Rta]):-
       flick(Grid,Color,FGrid),
       ayudaBasica(FGrid,L),
       max_member(CantSegunda,L),
       R is Cant + CantSegunda,
       ayudaAdicional2(Grid,Rs,Ls,Rta).

%Verifica si se dio una victoria mediante la grid que recibe.
%Retorna 1 en caso de victoria, 0 en caso contrario.
verificarVictoria(Grid,1):-
       flick(Grid,y,GridN),
       grid(5,GridN).
verificarVictoria(_,0).
```