Documentación

Requerimientos

• Se deberá permitir al jugador elegir entre por lo menos 4 configuraciones de la grilla precargadas.

• Cuando el jugador lo solicite se le brindara ayuda en el juego. Concretamente, para cada uno de los 5 colores posibles que el jugador puede escoger en la instancia actual del juego, deberá brindarse información de la cantidad de celdas que se sumaran al conjunto de celdas en relación adyacenteC* con la celda superior izquierda si se presionara el botón de dicho color. La cantidad de celdas incorporadas es un indicador (entro otros posibles) de la calidad de la jugada.

• El usuario podrá optar por obtener aun mayor ayuda, obteniendo para cada uno de los 5 colores la cantidad máxima de celdas que se sumarán al conjunto de celdas en relación adyacenteC* con la celda superior izquierda si se presionara el botón de dicho color, y luego se presionara un botón de otro color. Es decir, para cada color C, el beneficio máximo que puede llegar a obtener en 2 jugadas si elige como primera jugada C.

Decisiones de implementación:

• El modelado de la Grilla de 14x14 es a través del predicado dinámico celda/2, que mantendrá registradas las 196 celdas de la grilla en la base de datos durante toda la ejecución del programa.

• Nos valdremos del predicado dinámico visitado/1, que nos será útil al momento de recorrer la grilla.

• El predicado dinámico turno/1 funcionará a manera de contador, incrementándose cada vez que el jugador realice una jugada válida.

Implementación en Prolog

comenzar(?Int1):- Realiza una nueva jugada. Obtiene todos los adyacenteC* de la celda 1 en una lista, se eliminan los elementos repetidos de ésta y con la lista sin repetidos y el parámetro int1 invoca al método pintar/2. Aumenta un turno y desmarca todas las celdas que fueron marcadas como visitadas en el método adyacenteC/2.

aumentarTurno:- Aumenta en uno el contador con el que se lleva el control de turnos.

chequearVictoria(?Int1):- Obtiene todos los elementos de la grilla en forma de lista, y la usa como parámetro para llamar al método todosIguales/2. Resulta exitoso cuando todosIguales/2 retorna verdadero.

todosIguales(?List, ?Int1):- Es un método recursivo que resulta exitoso si todos los elementos de la lista List son iguales al parámetro Int1 que denota el color. En el caso recursivo se quita el primer elemento de la lista, se controla el color y se vuelve a llamar al método con la lista sin su primer elemento, termina cuando algún color no se comprueba o al llegar al caso base con la lista vacía.

adyacenteC(?Int1, ?Int2):- Busca la celda en relación de adyacenteC con la celda Int1, marcando como visitadas todas las celdas que va recorriendo. Retorna verdadero si la celda Int2 es adyacenteC a Int1.

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.

adyacente(?Int1, ?Int2):- Int2 es una celda adyacente a Int1. Diremos que una celda es adyacente a otra si se encuentra a la derecha, izquierda, arriba o debajo de ésta en la grilla.

pintar(?List, ?Int):- Recorre la lista List e invoca al método pintarAux/2 con cada elemento de la misma.

pintarAux(?Int1, ?Int2):- Elimina la celda Int1, y crea una nueva celda Int1 con color Int2.

cambiarGrilla:- En caso de ya haberse creado una grilla, la elimina, invoca al método crearGrilla/1, y actualiza el contador de turnos.

crearGrilla(?Int1):- Crea una nueva grilla de manera aleatoria, con Int1 celdas.

eliminarRepetidos(?List, ?List):- Recibe una lista “ListaIn”, elimina los elementos repetidos y los devuelve ordenados en la lista “ListaOut”.

ayuda(?Int1, ?Int2):- Retorna la cantidad de celdas que se sumaran al conjunto de celdas en relación adyacenteC* con la celda superior izquierda si se presionara el botón del color Int1. Para lograr ésto primero se obtienen en una lista todos los elementos adyacenteC a la primera celda, se guarda el largo de la lista en una variable "Size", luego se clona la grilla en su instancia actual para poder pintar del color Int1 a todos los adyacentesC conseguidos anteriormente sobre ésta última sin alterar la grilla que se está usando en el juego, en una lista se obtienen todos los adyacentesC desde la celda 1 pero de la grilla clonada y ya pintada, se guarda el largo de esta última lista en una variable "Size2" y luego se elimina el clon y se retorna en la variable Int2 la diferencia entre Size2 y Size.

adyacenteCClon(?Int1, ?Int2):- Busca la celdaAux en relación de adyacenteC con la celdaAux Int1, marcando como visitadas todas las celdasAux que va recorriendo. Retorna verdadero si la celdaAux Int2 es adyacenteC a Int1. Este método trabaja sobre un clon de la grilla que luego será eliminado.

pintarClon(?List, ?Int1):- Recorre la lista List e invoca al método pintarAuxClon/2 con cada elemento de la misma.

pintarAuxClon(?Int1, ?Int2):- Elimina la celdaAux Int1, y crea una nueva celdaAux Int1 con color Int2.

clonar:- Obtiene todas las celdas de la grilla en una lista, y funciona como cascarón para invocar al método clonar2/1 con dicha lista.

clonar2(?List):- Crea un clon de la grilla actual con celdas llamadas celdaAux.

ayuda2(?Int1, ?Int2):- Obtiene la cantidad máxima de celdas que se sumaran al conjunto de celdas en relación adyacenteC* con la celda superior izquierda si se presionara el botón de color int1, y luego se presionara un botón de otro color. Es decir, para cada color C, el beneficio máximo que el jugador puede llegar a obtener en 2 jugadas si elige como primera jugada C.

Lo primero que hace este método es averiguar la cantidad de celdas que se sumarán en el primer paso, es decir seleccionando como primera jugada C, para esto usa el método ayuda/2 y guarda esta cantidad en la variable Cant1 luego obtiene en una lista todos las celdas adyacenteC a la primera celda y guarda en la variable "ColorViejo" el color actual de todas esas celdas guardadas. Una vez guardado el color, se pintan todas las celdas de la lista del color Int1 y se llama al método mejorJugada/1 que nos devuelve en la variable "Max" cuál sería la mejor jugada en ese momento, luego se vuelve a pintar la lista de celdas que teníamos del "ColorViejo" y se retorna en Int2 el valor de "Cant1" + "Max".

mejorJugada(Max):- Llama al método ayuda/2 con cada color posible para determinar cuál sería la mejor jugada en esa instancia del juego, retorna el valor representativo del número en la variable Max. Utiliza el método comparar/2 para encontrar el mayor valor.

comparar(?List, ?Int):- Recorre los elementos de la lista List, y Int es el mayor elemento de dicha lista.

elegirGrilla(?Int):- Selecciona una de las 4 grillas precargadas disponibles para comenzar el juego.

Interfaz Java – Prolog

Implementamos 2 clases en Java: Selector y Game.

La clase Selector solo se encarga de setear parámetros para ser utilizados en ‘Game’. La clase ‘Game’ es la consulta métodos de Prolog desde Java.

Métodos:

Se utiliza el método comenzar/1 para dar comienzo a la jugada cuando el usuario hace click en alguno de los 6 botones de los colores. Prolog se encargara de modificar la

grilla de acuerdo a la jugada realizada, para que luego en Java actualicemos la interfaz gráfica para que se corresponda con la base de datos que nos brinda Prolog. Esto lo hacemos utilizando la consulta de Prolog ?-celda(X,Y), entonces al botón X de Java le asignamos el “color” Y.

Consultamos al método chequearVictoria/1 inmediatamente después de que comenzar/2 haya finalizado. El método verifica las celdas de Prolog, y si son todas del mismo color retorna verdadero. En caso de retornar verdadero Java muestra por pantalla un mensaje de victoria y finalizará la partida.

El método ayuda/2 es consultado automáticamente cuando el usuario marca la casilla Ayuda1 de Java. Este método recibe un color y un entero. El entero es la cantidad máxima de celdas que se agregaran a la grilla para ese color. Esto lo realiza Prolog. Desde Java llamamos al metodo ayuda/2 seis veces, una vez para cada color, y cada resultado que obtenemos lo mostramos por pantalla.

El método ayuda2/2 es consultado automáticamente cuando el usuario marca la casilla Ayuda2 de Java. Este método recibe un color y un entero. El entero es la cantidad máxima de celdas que se agregaran a la grilla para ese color y para otra jugada con otro color distinto. Esto lo realiza Prolog. Desde Java llamamos al método ayuda2/2 seis veces, una vez para cada color, y cada resultado que obtenemos lo mostramos por pantalla. Aclaramos que no se le informa al usuario cual es el segundo color para realizar la mejor jugada posible.

ElegirGrilla/1 lo utilizamos para que Prolog escriba en la base datos la grilla precargada indicada por el parámetro, para luego en Java poder recorrer toda la grilla y representarla gráficamente.

El método cambiarGrilla/0 lo utilizamos para que Prolog escriba en la base datos una grilla completamente aleatoriaj, para luego en Java poder recorrer toda la grilla y representarla gráficamente.