Proyecto de Prolog: Hive

Victor Manuel Cardentey Fundora C411
David Guaty Domínguez C412

Estructura del proyecto

El proyecto consta de 4 módulos:

1. app.pl: El módulo encargado de la interacción con el usuario mediante una interfaz gráfica realizada en pce.

2. board.pl: El módulo encargado de modelar la estructura actual de las piezas colocadas. Brinda información importante sobre el estado actual del juego.

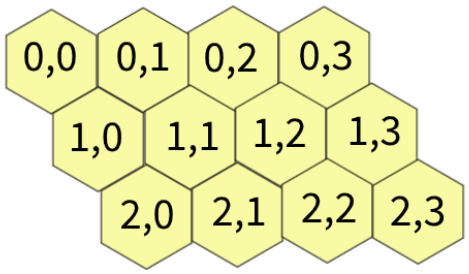
3. bugs.pl: EL módulo encargado del manejo de los distintos tipos de insectos. Puede calcular los posibles movimientos de cada insecto.

4. cpu.pl: El módulo encargado de implementar el jugador no humano.

board.pl

Modelado del tablero

Aunque el juego no presenta un tablero, el modelado de las posiciones de las piezas sitúa las piezas en un grid hexagonal infinito. Para modelar un grid hexagonal se tienen varias opciones, en este proyecto se eligió rotar el eje x del grid de tal forma que cada fila de hexágonos está desplaza a la derecha por medio hexágono con respecto a la anterior. Por ejemplo, un grid hexagonal de 4x4 se vería así:



Módulo board

Teniendo en cuenta la definición de grid hexagonal anterior, se implementó el módulo board para obtener información acerca del estado actual del juego.

Para saber si dos casillas son adyacentes se tiene el predicado adyacent(X1, Y1, X2, Y2), que triunfa si el hexágono con coordenadas X2,Y2 es adyacente al hexágono X1, Y1.

Para llevar el estado del juego, board.pl tiene varios predicados dinámicos que van cambiando de acuerdo a si se coloca una pieza nueva o si se mueve una pieza.

Para saber si en la celda con coordenadas X,Y existe un insecto en juego se tiene el predicado bug/5.

bug(C,T,X,Y, S) triunfa si existe un bug de color C(con color se hace referencia a un tipo de jugador), de tipo T(hormiga, reina,etc), que está en la posición X,Y. La variable S se refiere a la posición del insecto dentro de la pila de insectos que se puede formar, por ejemplo un escarabajo que esté encima de una reina tendrá valor S =1, mientras que la reina tendrá valor S=0.

A continuación se explica como se colocan piezas.

Celdas fronteras

Para colocar un insecto en juego, este tiene que ser adyacente a algún insecto de la colmena, si no se cumple esto se rompería la colmena. Por lo que se tiene la definición de **celda frontera**. Una celda vacía con coordenadas (X,Y) es **celda frontera** si es adyacente a una celda no vacía.

Para saber si una casilla está vacía se tiene: *empty(X,Y):- \+ bug(_,_, X, Y, _)* .

Manteniendo almacenadas las celdas fronteras y colocando insectos

Para evitar calcular todas las celdas fronteras cada vez que se vaya a colocar un insecto, se tiene el predicado dinámico frontier/2, frontier(X,Y) triunfa si la celda con posicion (X,Y) es frontera.

Para colocar un insecto en juego se tiene el predicado placeBug(C,T,X,Y): coloca el insecto con color C, de tipo T, en la posición X,Y.

Al colocar un nuevo insecto en la celda X,Y que es frontera, se expande la frontera de la colmena, almacenando así las nuevas casillas fronteras. En el placebug se tiene la siguiente línea: *forall(emptyAdyacent(X,Y,X1,Y1), setFrontier(X1,Y1))*. Tambien se setea un nuevo bug, o sea se añade el predicado bug(C,T,X,Y,0) en la base de datos.

Nota: El placeBug hace un poco más, porque también sirve para el movimiento de las piezas. El placeBug del módulo escoge la altura de la celda con el predicado getCellHeight(X,Y,H) donde se obtiene en H cuantas piezas hay apiladas en la posición X,Y. El insecto con mayor altura dentro de una celda con altura H es H-1. Luego coloca un bug en la altura H seteando bug(C,T,X,Y,H).

Moviendo insectos

Para saber si un insecto se puede mover se tiene que analizar si la colmena no se divide al remover la pieza del tablero.

Lema 1: Si un insecto no se puede remover de la colmena, entonces no puede moverse.

Suponga que el insecto a analizar está en la celda A. Se sabe que al removerlo la colmena queda desconexa. Eso significa que existen al menos dos celdas adyacentes a A, X y Y tal que al remover la pieza en A no hay forma de alcanzar a Y partiendo de X. Como dos celdas tienen a lo sumo un par de celdas adyacentes en común se consideraran los siguientes casos:

1. A es la única casilla en común con X y Y: En este caso a cualquier lugar al que se mueva el insecto en A quedarían desconexas las celdas X y Y.

2. Existe otra celda en común con X y Y: Sea la otra celda en común A1, si el insecto en A logra llegar a A1 entonces puede mantener la colmena conexa, pero note como el insecto no puede ir de A hacia A1, porque incumple la norma de que la ficha tiene que se deslizada, y tampoco puede ir a otra celda intermedia porque dividiría la colmena en su camino. ■

Luego de que se compruebe que el insecto seleccionado puede moverse, se calculan los posibles destinos a los que puede ir mediante el modulo bugs.pl.

Para mover una pieza se utilizan los predicados removeBug y luego placeBug(discutido anteriormente).

Removiendo insectos

Al remover un insecto se deben actualizar las celdas que dejan de ser fronteras. Las celdas fronteras que dejarán de serlo al remover el insecto serán las celdas **aisladas** que son adyacentes a la celda en la cual está el insecto a remover.

Una celda frontera es aislada si solo posee una celda adyacente no vacía.

Comprobando si la colmena es conexa

Para saber si un insecto en la celda X se puede remover se escoge una celda no vacía adyacente a X y se realiza un dfs a partir de ella sin pasar este por la celda X(para esto solo hay que poner en la base de datos que se visitó la celda X antes de hacer el dfs). El dfs va insertando en la base de datos un predicado visited(X1,Y1) que triunfa si el dfs ha pasado por esa celda. Al final se comparan la cantidad de celdas visitadas con la cantidad de celdas no vacias, si son iguales entonces el insecto se puede remover y la colmena seguirá siendo conexa.

El tablero también guarda otras variables que modelan el estado del juego, como por ejemplo la cantidad de piezas que cada jugador posee en la mano, el último insecto jugado, entre otras.

bugs.pl

Es el módulo encargado de analizar los posibles destinos de las piezas.

Celda Accesible

El concepto de celda accesible es el que permite reconocer las posibles movimientos de la mayoría de las piezas: las piezas que se deslizan.

Se tiene el predicado accessibleCell(X1, Y1, X2, Y2) que triunfa si la celda X2, Y2 es accesible desde la celda X1,Y1 en un movimiento.

Una celda X2,Y2 es accesible desde X1,Y1 si X2,Y2 es una celda frontera adyacente a X1,Y1 y además X2,Y2 y X1,Y1 comparten una casilla vacía adyacente, esto último es para asegurar la condición de que las piezas tengan que deslizarse. Además se comprueban otros cosas para casos extremos como que la pieza trate de moverse hacia una **celda aislada**.

Para obtener los destinos de cada pieza se supone que pueden ser removidas, el concepto de celda accesible asegura que la pieza se mueva siempre por la colmena sin dividirla en dos.

Reina

Para la reina, todos sus posibles destinos son simplemente las celdas accesibles a ella.

Escarabajo

El escarabajo puede moverse a sus celdas accesibles o cualquiera de las celdas adyacentes a el que no estén vacías.

Saltamontes

Para analizar los posibles destinos del saltamontes, para cada celda adyacente no vacía a él, se realiza lo siguiente: Sea la celda del saltamonte X,Y y la celda adyacente no vacía X1, Y1. Se forma el vector V(X1 - X, Y1 -Y) y se avanza en esa dirección hacia X3,Y3. Si X3,Y3 está vacía, X3,Y3 se cuenta como posible destino, si no se repite el proceso hasta encontrar una celda frontera.

Araña

Para la araña se programa los posibles destinos por la definición. Dada la posicion X1,Y1 de la araña se encuentran (X2,Y2),(X3,Y3),(X4,Y4), tal que accesible($X_i, Y_i, X_{i+1}, Y_{i+1}$), i= 1,2,3 y además todas las celdas sean distintas.

Hormiga

Para la hormiga se hace un dfs que va agregando a la base de datos el predicado dinámico destination(X,Y) que triunfa si el dfs ha visitado la celda X,Y. El dfs visita una celda si esta es accesible desde la celda actual en la que se está analizando el dfs.

Mariquita

Al igual que con la araña se procede por definición. Dada la posición de la mariquita X1,Y1 se encuentran (X2,Y2), X3,Y3), (X4,Y4) tal que X2,Y2 sea adyacente no vacía a X1,Y1; X3,Y3 sea adyacente no vacía a X2,Y2 y X4,Y4 sea adyacente vacía a X3,Y3.

cpu.pl

El módulo cpu contiene una implementación del algoritmo minimax para obtener la próxima jugada. Mediante varias métricas y pesos asociados a dichas métricas se forma una heurística para evaluar un estado del juego y determinar quien tiene la ventaja. Para optimizar la búsqueda se realizó el pruning Alpha-beta, en el cual se mantienen dos valores, uno almacena la mínima puntuación que el jugador que maximiza tiene asegurado(alpha) y otro almacena la mayor puntuación que el jugador que minmiza tiene asegurada(beta). Si en algún momento beta $<$ alpha el nodo actual que se esté analizando no necesita expandirse más, porque el adversario puede forzar una jugada que resulte en un peor desempeño que el obtenido hasta ese momento.