Informe Escrito Proyecto : Hive

Programación Declarativa

Integrantes

Amanda González Borrell C411.

Karla Olivera Hernández C411.

El proyecto nuestro se divide en varios módulos: main, visual, move, place, ia ,utils, dynamic el principal es el main.pl el cual llama a los siguientes:

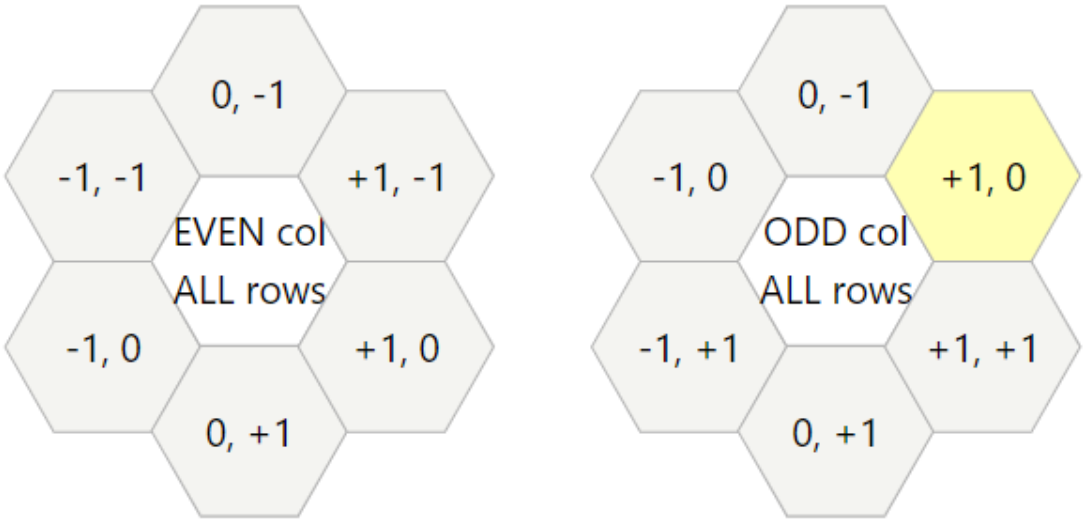
Se debe ejecutar:

```
[main].
#Se llaman a las  restantes partes del juego:
```

dynamic.pl: Aquí colocamos todos los predicados dinámicos para conservar el estado del juego .

- Los predicados principales son *chip(P, T, C, R, K)*: jugador (P), tipo (T), columna (C), fila (R), identificador (K). el cual se agregara cuando una ficha se agregue al tablero , los tipos son todos los especificados en la orientación del proyecto . el K es un identificador que hace a la ficha única, el P es el Player /0,1 y C y R la posición en el tablero .

  Para poder crear el tablero formado por hexágonos regulares utilizamos la siguiente forma:



El primer índice es la columna y luego la fila y se diferencia en si la columna es par o impar entonces ,las casillas adyacentes de una ficha en el tablero depende de que si la columna es para o es impar si debo sumar , restar 1 o mantener el mismo numero.

- *current_player(P)*: jugador (P). Cada vez que se cambia un turno o sea que le toca jugar a un jugador se elimina el predicado y se agrega uno nuevo con el numero del player /0,1.
- El juego de se basa en mover fichas por tanto necesito saber si existe un camino al cual se pueda mover una ficha por tanto utilizamos el predicado *mark(C, R, D)*: columna (C), fila (R),

distancia (D), el cual marca una posición(C,R) si la visito empezando en una posición Ci,Ri que esta a distancia.

- El predicado *above_of(P, T, C, R, N, K)*: jugador (P), tipo (T), columna (C), fila (R), nivel (N), identificador (K), se utiliza para aquellos insectos que pueden terminar su movimiento encima de la colmena , escarabajo, mosquito,.. , este predicado indica q la ficha que esta en la posición C,R con índice K del player P esta a nivel N o sea que si esta a nivel 1 es que esta encima de una ficha , nivel cero no tenemos pues en este caso el predicado no hay necesidad de incluirlo en la base de datos pues ya sabemos mediante chip que hay una ficha en esa posición.
- *transformation(P, K)*: el mosquito del jugador P con id K está transformado en un escarabajo, se utiliza para poder transformar a un mosquito es escarabajo cuando este aun no se ha bajado de la colmena .
- *movement(P, K, 0)*: la ficha con identificador K es la última que movió el jugador P. *movement(P, K, 1):* la ficha con identificador K es la última que movió el pillbug del jugador P.
- *board(W)*: guarda la referencia del tablero en el visual.
- *draw(H)*: H es un hexágono pintado de una posición disponible.
- *drawchip(C, R, K, H, F) *:- la ficha con identificador K que se encuentra en la posición (C,R) está dibujada y H y F son las referencias del hexágono y de la figura respectivamente.
- *double(L)*: guarda una lista L Se utilizara para poder guardar información necesaria.
- *game_mode(0)*: el juego se encuentra en modo jugador contra IA.
- *game_mode(1)* : el juego se encuentra en modo IA contra IA.
- *best_quality(Q, J, L)*: la mejor calidad de los tableros que se generan.

Visual.py : en el cual tenemos toda la parte de la programación de la interfaz gráfica , se utilizó Xpce para crear la misma.

La interfaz gráfica esta formada por tres paneles el board q es el tablero y dos paneles de fichas blancas y negras ,tenemos dos modos de juego el player contra la inteligencia artificial e inteligencia artificial contra inteligencia ,para decidir cual modo desea activar debe escoger una de las opciones mostradas en pantalla y presionar reiniciar , de no presionar el modo activado por defecto es el modo jugador contra inteligencia artificial. El jugador es las fichas blancas al iniciar el juego escogerá que ficha ubicar en el tablero y se mostrará la posición en que puede ser ubicada, luego se pasará el turno a la IA.

La parte de Dibujar esta divida para la inteligencia Artificial y para el player ya que el player debe tener eventos para poder interactuar con el visual, estos son , tocar una ficha y que te pinte las diferentes posiciones de movimientos de ubicación y luego tras tocar esta posición , la ficha se mueva de la inicial a la final donde presionó clic y en lA solo hace falta mover la ficha de una posición a la otra ya sea por movimiento o ubicación . En el modulo visual también tenemos todas las cláusulas que regulan el funcionamiento del juego, dígase *start* que inicia el juego : dibuja el tablero, inicializa los predicados necesarios en la base de datos, entre otros, *restart* que lo reinicia, *game_over* que es cuando la reina de un jugador esta completamente bloqueada o sea no tiene un adyacente vacío al cual moverse , *end_of_turn* , cambia el jugador y en caso de que le toque jugar a la IA , llama a jugar a la IA .

En este módulo tenemos todo lo de dibujar las fichas, ya sea solo el hexágono para pintar las posiciones a las que se puede mover o ubicar o el hexágono con la imagen de la figura encima.

En el módulo *utils* están algunas clausulas generales que son utilizadas en varias partes de la lógica del juego, cada una de estas tiene un comentario en el código en donde se explica su funcionamiento.

En el módulo *move* están todos los métodos relacionados a los movimientos de los diferentes insectos .

Cabe aclarar que para evitar complejidad en el proyecto tenemos la siguientes pautas el player que seria la interacción del usuario con el visual, tiene sus predicados de generar movimiento que devuelve los movimientos válidos para una ficha que esta una posición inicial, tiene su predicado para mover una ficha ya que no es lo mismo generar los movimiento que mover la ficha ya que generar movimientos no es nada permanente simplemente devuelve los posibles lugares a los cuales se puede mover una ficha y el mover si guarda en la base de datos la nueva posición de la ficha, por eso cuando se llama a *move* en en ciertos lugares donde ya se sabe que lo que se tiene es un movimiento válido estos predicados son *generate_movement* , *do_movement* y estos predicados llaman a los predicados principales de mover una ficha y de generar movimiento que están en el módulo *move* y se hace esta diferencia porque el player necesita utilizar la información que brindan los métodos principales antes mencionado y con esta información realizar ciertas actividad, pinta las posibles posiciones finales producto de realizar movimientos, cambiar una ficha de lugar producto de un movimiento final, lo mismo pasa con la Inteligencia artificial que esta no se comporta de la misma manera del player por lo que debe tener su generar movimiento y ejecutar el movimiento como predicado propio de ella pero también utiliza los principales .Esto también ocurre para el ubicar una ficha en el tablero .

El hilo del juego para player contra IA es el siguiente : empieza el jugador 0 que es el player , como no hay fichas ocupadas debe tocar una ficha blanca del panel, al pintar estas ficha se le agregó un evento que al tocarlas se llamará a la cláusula *generate_place* el cual llama a *places_available()* el cual genera todos los movimientos posible para esa ficha ,llenado una lista de posiciones validas, para una ficha primero se verifica que el el tipo de ficha que quieres poner ya sea escarabajo, saltamontes,no puede exceder a la cantidad disponible para saber cuantas fichas de un tipo tiene cada player tenemos una clausula ,al igual para saber cuantas fichas tiene un player ubicada *(total_player(P, T, N) ,total_chips(N))* respectivamente, si a *total_player* solo le pasas como argumento el player te da l cantidad de fichas de fichas en el tablero de un jugador y si la pasas el Player y el T te retorna la cantidad de fichas de un tipo de un jugador , total_chip nos permite también verificar que al ubicar una ficha si es la 4ta q ubico no puedo ubicarla a no ser que esta sea la oveja reina. Al generar las posiciones en las que puede ubicarse la fichas estas son aquellas en las que al colocarla la ficha este conectada , como yo en cada ubicación garantizo que todas las fichas estén conectadas ,cuando se ubica una nueva solo debo verificar que tenga a un adyacente no vacío pero como una ficha solo puede tocar a piezas de su mismo color en la ubicación excepto la segunda, entonces debemos verificar que este adyacente sea del mismo player(color) y esto debe cumplirse para todos los adyacentes las funciones las funciones *same_player, valid_chip, valid_queen* situadas en el módulo place describen si una posición es válida o no . Luego de generar las posiciones válidas se deben pintar, para esto esta la clausula place si al tocar una ficha y tocar las posiciones válidas pintadas en el tablero se llamara *draw_possible_places* que pintara estas posiciones. Cuando pulsas sobre una posición válida para pintada para ubicar llamas a la cláusula *draw_placement* que se encarga de llamar a *place* que agrega la ficha a la base de datos y luego la pinta. Al pintar la ficha en el tablero en su posición inicial le crea un evento para cuando la toques llamar a la cláusula *generate_movement()* el cual se encargara de generar los movimiento para mover la ficha que tocaste, pero antes ,luego de pintar la ficha se llama a *end_of_turn* para verificar si con la jugada que hizo perdió o ganó cambia al próximo jugador, q es la IA en este caso lo que hacemos después es llamar a play_ia(1) para que juegue, como juega la IA lo explicaremos a continuación.

Inteligencia Artificial

La inteligencia para un estado del tablero genera todos los posibles movimientos de las fichas ubicadas y las posibles posiciones iniciales en las que puede entrar una ficha al juego. Por tanto, hay dos posibles jugadas: J(0) si se decide ubicar una ficha en el juego y J(1) si se decide mover una ficha. Entre todas las posibles jugadas, la inteligencia artificial se queda con la que mejor calidad posea. Cuando se habla de calidad nos referimos a la siguiente heurística:

- Si la reina adversaria tiene a alguien encima, suma unos 20 puntos.
- Se suma la cantidad de fichas que rodean a la reina adversaria menos cantidad de fichas que rodean a la mÃa, multiplicado por 10.
- Se suma la cantidad de fichas que no puede mover el adversario menos cantidad de fichas que yo no puedo mover.
- Si en ese estado del tablero se gana, suma 150 puntos.
- Si en ese estado se pierde, resta 150 puntos.
- Se suma la cantidad de fichas en el campo menos la cantidad de fichas que aún no se han colocado.
- Se suman la cantidad de hormigas del adversario que se encuentran bloqueadas, ya que estas fichas son las que mas libertad de movimiento tienen y son las que más interesa que no se puedan mover.

Esta heurística fue implementada en el predicado *board_state(P, Q)*.

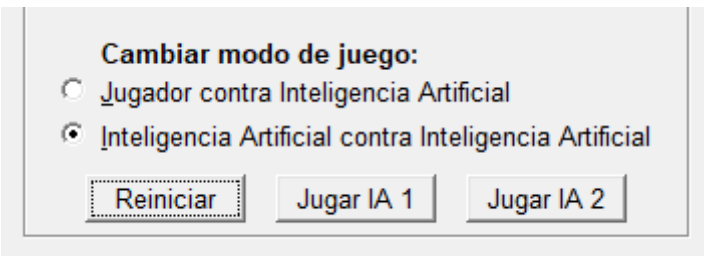
Para evitar que la IA siempre realice las mismas jugadas, en caso de tener dos jugadas o mas con la misma calidad decidimos de forma aleatoria entre todas ellas con cual quedarnos. Para esto, cada vez que se llama a generar una jugada de la IA, se crea un predicado que a medida que se analice un estado del tablero si tiene una calidad mejor a la del predicado, se elimina este predicado y se agrega uno nuevo con la calidad y una lista que contiene la jugada nueva; en caso que sea igual se elimina el predicado y se aÃ±ade uno nuevo que contiene las jugadas anteriores y la nueva, manteniendo la misma calidad.

Luego de finalizar todas las generaciones en la base de datos encontramos en el predicado *best_quality(Q, L)* la mejor calidad Q encontrada y una lista L con las jugadas que la generaron. Los elementos de esta lista son de la forma: $[0, [T, C_i, R_i]]$ si se aÃ±ade una ficha nueva al tablero y $[1, [C_i, R_i, C_f, R_f]]$ si es un movimiento.

Las generaciones se realizan con ayuda de los siguientes predicados: *generate_all_places(P, L)* que unifica a L con la lista de todas las posiciones disponibles y los tipos de fichas disponibles que el jugador P puede añadir, *generate_all_mov(P, L)* que unifica a L con los posibles movimientos de las fichas puestas en el tablero y un predicado especial *generate_all_pb_lifting(P, L)* , el cual si el jugador P ya ubica al pitbull en el tablero, entonces genera los posibles movimientos que pueden hacer sus adyacentes colocandose encima de l y luego bajando a una posición vacía. Cada lista L unificada en los predicados anteriores, es recorrida usando los predicados: *for_list_plac(P, L)*, *for_list_move(P, L)* y *for_list_lifting_pb(P, L)* respectivamente. Dentro de estos se llaman a los predicados: *analyze_place(P, T, C, R)*, *analyze_move(P, Ci, Ri, Cf, Rf)* y *analyze_move_pb(Ci, Ri, Cf, Rf)* respectivamente, los cuales llaman al predicado *update_quality(Q, J, L)* que se encarga de hacer la actualizaciÃ³n correcta del predicado *best_quality(Q, L)*.

Luego de terminar de llamar a todos los predicados explicados anteriormente, es hora de escoger la juagada que va a realizar la IA, la cual es escogida de manera random entre todas las mejores jugadas guardadas. Al obtener una, se ejecuta llamando a place o a move segun haya sido la escogida y pintamos el tablero.

Para ejecutar el modo IA contra IA, en la aplicación contamos con el siguiente panel:



Para comenzar el modo de juego IA contra IA, se selecciona la segunda opción y se presiona el botón de *Reiniciar*. Luego para que cada IA realice su movimiento se va presionando el botón correspondiente *Jugar IA*.

Move

Aquí explicaremos los movimientos de todas las fichas ubicados en el módulo move, los métodos del player y de la Ia llaman a estos ya lo explicamos anteriormente.

Para generar los movimientos de una ficha se necesita su posición inicial , el player, el tipo , lo primero que se debe verificar es si esta ficha puede moverse para ello verifica si ella es el tope de la colmena en su posición para esto utilizamos el above_of predicado dinámico ya que este si existe una ficha que esta encima de otra existe tal predicado con el nivel en q se en encuentra en caso de que el predicado no exista solo hay que verificar que el predicado chip con los datos de la ficha inicial están .

generate_mv es el predicado principal.

Empecemos por los insectos principales del juego , abeja reina, escarabajo, saltamontes, araña ,hormiga,

Como en todo momento la colmena debe estar conectada o sea no puede dividirse en dos lo primero que verificamos para todos es que si eliminado la ficha que unifica con la posición inicial de la ficha que se quiere mover no desconecta las fichas del tablero ,las cuales podemos ver como nodos de un grafo,al final de la generación de movimientos hay que volver a agregar la ficha. Para verificar q este grao esta conectado tenemos una clausula graph_is_connected(), en el cual comienza a marcar desde una posición del grafo todos aquellos adyacentes a los cuales puede llegar marcando a través del predicado dinámico mark(C,R,_) , eliminamos los duplicados y verificamos que sea la cantidad de fichas total, entonces como esa ficha estaba eliminada y el grafo no se desconecto la movemos de ahí para otro lugar y el grafo continuara en ese estado si la posición final al menos tiene una ficha adyacente.

Como todas las fichas para moverse no pueden desconectar al grafo y están encima de la colmena después de generate_mv se pasa a generate_mv2 y generate_mv1 que hacen lo anterior y a partir de generate_mv1 se harán pasos distintos para cada tipo , para lograr generar los movimientos .

En la reina luego de saber que puede moverse ,falta verificar que se mueva a una posición valida o sea tiene que estar vacía : para lo que buscamos si existe un predicado chip con la fila y columna final , la posición debe estar conecta al grafo : buscamos si existe una ficha adyacente y por último debemos verificar que en la dirección en que se ejecutará el movimiento de la reina ella pueda pasar por ahí o sea que la posición final no este bloqueada, para verificar esto se utiliza la función is_not_blocked(Ci, Ri, D) el cual con una dirección verifica que las casillas adyacentes que quedan a los lados cuando te mueves en esa dirección , no estén ocupados a la vez. si todo lo anterior se cumple entonces es un movimiento válido para la reina.

Para el caso del Saltamontes este se mueve en línea recta hasta encontrar una posición vacía , y esta no debe ser una casilla adyacente de el para ello nos definimos un predicado que dada una posición fila , columna inicial le sume uno o reste 1 o mantenga igual la fila y la columna de modo que se mueva en línea recta hasta llegar a una posición vacía , llamamos a movernos en todas las posiciones del grafo a partir de la inicial pero una vez que empiece a recorrer solo puede moverse en esa misma dirección esto hasta que encuentre una posición vacía que se que existe,estas posiciones vacías son los posibles movimientos.

Para el caso del escarabajo este se mueve un solo paso pero puede subirse encima de la colmena sobre cualquier ficha , los movimientos del escarabajo es hacia todos sus adyacentes y si hay alguien en uno de los adyacentes finales pido al que esta en la cima . Lo importante está en el mover la ficha escarabajo hacia las posibles posiciones generadas ya que si hay alguien en esa posición debo incluir el above_of con el nivel en que se va a encontrar esta ficha que esta dado por la cantidad de fichas + 1 que hay ahí en esa posición y cuando baje de la colmena elimino el above_of que tenga como id y fila, columna la de la ficha del escarabajo que se movió.

La araña y la hormiga se mueven de manera similar alrededor de la colmena bordeándola, por tanto las dos se basan en una idea general para generar posiciones válidas de movimiento. A partir de una posición inicial me muevo por las casillas adyacentes marcándolos con mark y con su argumento D profundidad que es la distancia mínima a la que tu estas de mi , si una posición no esta marcada , esta vacía , conectada al grafo y no esta bloqueada, la marco con profundidad D y llamo a marcar los adyacentes con D + 1 si la casilla esta marcada entonces verifico que la D que tiene el marcado no sea mayor que la D con la que vengo a marcarlo si es así la actualizo , y en caso de que la distancia con la que se marco sea menor que la distancia con la que yo vengo a marcarlo cambio la distancia con la que voy a seguir marcando a los otros y verifico que la distancia de los adyacentes de este marcado que lo estaba con menor distancia que yo estén bien si no es el caso la actualizo . Las posiciones válidas son todas aquellas marcadas. La diferencia entre la araña y la hormiga es que una solo son las marcadas con distancia tres y la otra tiene libertad de número de fichas que puede moverse.

Continuamos con las expansiones

La mariquita lo que hacemos es visitar los adyacentes de la posición inicial y luego a sus adyacentes siempre y cuando la distancia sea menor que tres y estas casillas estén ocupadas por fichas y la casilla con Distancia tres debe estar vacía, por tanto si encontré un camino con distancia tres que cumple con lo anterior la posición final es valida .

El mosquito se mueve copiando el movimiento de alguno de sus adyacentes por tanto como para todas las fichas debemos eliminarlas para verificar si estas conectado lo que hacemos es agregar nuevamente la ficha la del mosquito con su mismo id, fila y columna pero con el tipo de un adyacente , genero el movimiento para el y todas sus posiciones finales son válidas para mi, luego elimino esta ficha del tablero, lo importante de esto es que en el mover de una posición a otra el mosquito si este se transformó en un escarabajo y se quedó encima de la colmena debo seguir siendo un escarabajo por tanto no elimino la ficha con tipo escarabajo simplemente agrego un predicado a la base de datos transformation con el id de la ficha del mosquito guardada como escarabajo para cuando ese escarabajo baje de la colmena con un movimiento saber que ese Id era el de un mosquito y volver a su estado de mosquito, eliminado el escarabajo y agregado el mosquito.

El bicho bola se mueve igual que la reina pero puede decidir si moverse o mover a uno de sus adyacentes, por tanto sus posiciones finales son los adyacentes vacíos , conectados al grafo y no bloqueados , y las posiciones finales de los adyacentes si decido no moverme y moverlos son los adyacentes vacíos del bicho bola ya que el mismo sube a los adyacentes sobre el y luego los baja a una posición vacía. En el *move* hay que tener en cuenta que el bicho bola no puede mover la ultima pieza que movió el adversario esto lo sabemos con el predicado *movement(P, K, 0)* con el P de adversario si la ficha con K que yo quiero mover es la misma que la K del *movement* no puedo moverla, y tampoco se puede mover la pieza movida por un bicho bola y esto se logra también con *movement(P, K, 1) * pero con el ultimo parámetro en 1 que se agrega cuando un bicho bola mueve a un adyacente y se elimina cuando pasa el turno del adversario , si la ficha que quiero mover tiene ese índice k no puedo moverla . Falto mencionar que todos los adyacentes no son candidatos a moverse , estos deben cumplir que al quitarlos de esa posición ellos no desconectan al grafo ,además el bicho bola no podrá mover un adyacente que este apilado o apile a alguien o

sea que si hay mas de una ficha en la casilla adyacente no puedo moverla, también las posiciones finales no solo tienen que estar vacía sino también no pueden estar en un hueco con piezas apiladas , por tanto para nosotros un huevo con piezas apiladas es aquel que la pieza esta vacía en el medio y los adyacentes están ocupados con fichas y todos deben tener mas de una ficha encima.

Falto mencionar que en generar los movimiento y mover diferenciamos al bicho bola de los demás insectos porque lo información que devuelve no es la misma , lo mismo pasa con el pintar , para cualquier otra ficha e pintar los movimientos y luego mover la ficha a la posición que seleccionó , en el bicho bola también necesito pintar los posibles adyacentes .