Proyecto de Programación Declarativa: Juego Hive

♣ Darian Dominguez Alayón C-411

♣ J.Alejandro Campos Matanzas C-411

Facultad de Matemática y Computación

Universidad de La Habana

Curso 2020-2021

Para la resolución de nuestro proyecto creamos una parte lógica y una parte visual. A continuación explicaremos como funcionan cada una de ellas:

Lógica

En nuestra lógica tenemos la siguiente estructura:

1. **hexagon**: contiene todos los predicados relacionados con la lógica y geometría de una casilla.

2. **insects**: aquí están presentes todos los predicados relacionados con la lógica de los insectos y además es quien inicializa los valores.

3. **ia**: contiene predicados que son de ayuda en el funcionamiento de la IA.

4. **utils**: son predicados comunes que son usados por varias bibliotecas.

Explicaremos a continuación como funcionan estos módulos:

hexagon: Lo primero que hacemos es crear un predicado *direction* para definir las seis direcciones del hexágono. También tenemos los predicados *are_neighbors, are_neighbors_dir, neighbor_dir* que se utilizan para saber si dos casillas son vecinas y también podemos usarlos de tal forma que dado una casilla nos devuelva sus vecinos. Para el recorrido por las casillas creamos un predicado *dfs*, el cual es usado en los predicados *is_connected_hex_to_hive* y *connected* para ver si una casilla está conectada en la colmena; y *not_articulation_point* que nos dice si una casilla no es punto de articulación, o sea, que si la quitamos, la colmena sigue conectada. Tenemos además los predicados *flat_hex_to_pixel* y *pixel_to_flat_hex* utilizados para convertir de coordenadas axiales a pixeles y viceversa.

insects: Para trabajar con los insectos, creamos los predicados dinámicos: *insect* que contiene el tipo *Type*, id *Id*, jugador *Player_id*, hexágono *Hex=[R,Q]* y nivel *Level* (si un insecto tiene Level = -1, entonces está en la mano del jugador); *hive* que contiene todas los insectos de la colmena; *player* que contiene el id *Id*, número de movimientos *numb_move*, jugador actual *current* e inicio de jugada *init_play*; y un predicado *add_queen* para saber si debemos agregar la reina o no.

Tenemos el predicado *move_insect* que recibe el tipo de movimiento que se va a realizar (init, add, Type_insect), las características del insecto a mover y la casilla a la cual se quiere mover. Este predicado llama a su vez a *move_insect_db* que es el encargado de actualizar la base de datos de prolog respecto a los insectos y las posiciones.

Otros predicados son *find_insect_high_level*, que devuelve el insecto con nivel mayor en una casilla; *insect_blocked* que devuelve true si un insecto está bloqueado; *change_player_turn* devuelve true si después de una jugada el insecto a jugar tiene movimientos posibles; *queen_in_game* devuelve true si la reina del jugador está en juego; *must_add_queen* devuelve true si es el cuarto movimiento y todavía no se ha agregado la reina a la colmena; *end_game* y *tie_game* para chequear si un jugador ganó o hubo empate respectivamente; *start_insects* el cual para cada jugador inicializa (usando assert) los insectos con sus respectivas posiciones; *cell_in_center*, que devuelve true si desde una casilla [X1,Y1] no se puede acceder a una casilla [X2,Y2] pues no cabe por el espacio.

Para cada insecto tenemos un predicado para sus posibles movimientos y estos son usados en un predicado *possible_moves* que recibe un valor *Val* el cual puede se init: si es un movimiento inicial, add: si se va a gregar un insecto a la colmena y type_insect que es para llamar a los posibles movimientos de dicho insecto; además de recibir *Val*, recibe las características del insecto a mover y la posición del hexágono al cual se quiere mover.
Para el movimiento *init* devolvemos los posibles movimientos iniciales que serían para el primer jugador el [0,0] y para el segundo cualquier adyacente a [0,0].
Para el movimiento *add* de un insecto miramos todos los insectos que hay en la colmena que son del mismo color que él y devolvemos como posibles movimientos sus adyacentes vacíos, siempre y cuando estos adyacentes estén conectados a la colmena y no tengan ningún adyacente del color del otro jugador.
Para el movimiento *abejaReina* primero miramos si no está bloqueada, o sea, si no existe un insecto en la colmena en la misma posición y con mayor nivel, luego si no es punto de articulación y después miramos por cada casilla adyacente vacía si está conectada a la colmena que no contiene la abejaReina.
Para el movimiento *hormiga* miramos si no está bloqueada y no es punto de articulación, luego a través del predicado *hormiga_move* que simula un dfs vamos quedándonos con todas las casillas vacías que están conectadas y no cumplen que están en el centro, en la colmena que no contiene a dicha hormiga.
Para el movimiento *saltamonte* miramos si no está bloqueada y no es punto de articulación, luego tomamos la colmena que no contiene esa casilla y le aplicamos el predicado *saltamonte_move*, el cual analiza por cada vecino del saltamonte si hay una casilla vacía después de un conjunto X de casillas ocupadas.
Para el movimiento *aranha* miramos si no está bloqueada y no es punto de articulación, luego tomamos la colmena que no contiene esa casilla y le aplicamos el predicado *aranha_move*, el cual a partir de la posición de la aranha hace un dfs por niveles y por casillas vacías que estén conectadas a dicha colmena y devuelve true si en el nivel 3 la casilla está vacía.
Para el movimiento *mosquito* miramos si no está bloqueada y no es punto de articulación, luego tomamos la colmena sin el mosquito y miramos el tipo de movimiento de los adyacentes al mosquito y luego tomamos todos los movimientos posibles a partir de la casilla del mosquito con cada uno de los tipos de movimineto de los adyacentes.
Para el movimiento *mariquita* miramos si no está bloqueada y no es punto de articulación, luego tomamos la colmena que no contiene esa casilla y le aplicamos el predicado *mariquita_move*, el cual a partir de la posición de la mariquita hace un dfs por niveles y por casillas ocupadas que estén conectadas a dicha colmena y devuelve true si en el nivel 3 la casilla está vacía.
Para el movimiento *bichoBola* miramos si no está bloqueada y no es punto de articulación, luego tomamos la colmena que no contiene esa casilla y devolvemos las casillas vacías y conectadas que estén a su alrededor y si queremos aplicar la hablidad especial entonces miramos las casillas ocupadas y conectadas que estén a su alrededor, las cuales se van a poder mover para alguno de los espacios vacíos y adyacentes del bichoBola.

ia: Para la creación de la IA tomamos un enfoque greedy, el cual analiza en cada movimiento que la casilla a moverse esté lo más cerca posible de la abejaReina del contrario y lo más lejos posible de la abejaReina suya. Para ello usamos los siguientes predicados: *axial_distance* que nos da la distancia entre dos casillas; *find_move_more_far_to_queen* el cual devuelve la casilla [X1,Y1] más alejada a una casilla [X2,Y2] (usamos como [X2,Y2] la abejaReina de la IA); *find_move_more_near_to_queen* el cual devuelve la casilla [X1,Y1] más cercana a una casilla [X2,Y2] (usamos como [X2,Y2] la abejaReina del contrario).

utils: Aquí usamos alguno predicados que van a ser comunes para usar en otros módulos. Estos predicados son *bigger* que devuelve si un valor es mayor que otro; *remove_repeated* que devuelve una lista sin elementos repetidos; *delete_one* que elimina un elemento de una lista pero deja las repeticiones; switch que dado un valor manda a llamar al predicado que empieza por este valor; *rang* que devuelve los números en un rango determinado; *element_at* que devuelve el elemento i de una lista.

Visual

En nuestro visual tenemos la siguiente estructura:

1. **visual**: Es el encargado de ejecutar toda la parte visual en la cual nos auxiliamos de la librería pce.

2. **draw_visual**: Contiene los predicados usados para dibujar en el tablero

3. **utils_visual**: Contiene métodos útiles para ejecutar en el visual

Explicaremos a continuación como funcionan estos módulos:

visual: Aquí tenemos varios predicados dinámicos que nos van a servir en el desarrollo del visual como son: *bool_selected/1*, *piece_selected/5*,*pieces/4*, *init_player/2*, *move_state/1*, *current_player/1*, *arrowLeft/3*, *dimensions/3*, *dimensions_hive/4*, *dimensions_static/3*, *game_over/1*, *tie/1*, *win/1*, *message_end_game/1*, *last_piece_played/2*.
El predicado *main* es el encargado de ejecutarlo todo, estableciendo las posiciones de las fichas, creando la ventana del juego, el evento click y además pintando el tablero inicial.
Tenemos un predicado *draw_game* que auxiliándose de los predicados del módulo *draw_visual* lo que hace es dibujar el tablero.
Usamos también predicados para pintar y liberar botones. Estos botones son: los de la IA, los cuales si los pulsamos ejecuta el movimiento que haría nuestra IA; los botones de las direcciones, que son para mover el tablero por si las fichas no caben en pantalla (notar aquí que existe un límite de posición para poner las fichas en el tablero, el cual está dado por el rectángulo que crean las dos líneas trazadas bajo y sobre las fichas iniciales del player 1 y player 2 respectivamente); y además hay un botón para resetear el juego el cual llama al predicado *restart* y reinicia todos los valores.
Existe el predicado *draw_possible_movements* que es el encargado de dibujar los posibles moviminetos de una casilla a mover.
Con el predicado *click* es que controlamos primero si una casilla no está marcada y si es así la marcamos; si está marcada entonces se mueve la casilla marcada hacia la posición del click, siempre y cuuando la posición del click esté entre los posibles movimientos.
Con el predicado *move_piece* es que movemos las fichas de un lugar a otro en el visual.
Ahora como habíamos dicho anteriormente tenemos dos botones para ejecutar el movimiento que haría la IA y para ello nos apoyamos en los predicados siguientes: *get_possible_moves_initials* que devuelve todos los posibles movimientos de las fichas iniciales y *get_possible_moves_in_hive* que devuelve todos los posibles movimientos de las fichas en la colmena. Ahora solo nos queda elegir que casilla mover y hacia donde con nuestra heurística que se basa en un greedy. Para aplicar lo anterior tenemos seis predicados: *init_move_IA* que mueve la IA en el inicio; *must_add_queen_IA* agrega la reina de la IA si hay que agregarla; *add_init_IA* agrega las piezas si todavía no se pueden mover; *not_queen_IA_yep_queen_other* agrega las piezas de la IA en función de que la reina del contrario está agregada; *yep_queen_IA_not_queen_other* mueve las piezas de la IA en función de que la reina de la IA está agregada y la del contrario no; *yep_queen_IA_yep_queen_other* mueve las piezas de la IA en función de que la reina de la IA está agregada y la del contrario también.

draw_visual: En este módulo creamos los predicados siguientes: *draw_image_hexagon/4*, *draw_hexagon_axial/8*, *draw_hexagon_pixel/5*, *draw_hexagon_pixel_axial/5*, *draw_possible_movements/4*, *draw_pieces/8*, *draw_hexagon_pixel_e* *draw_hexagon_pixel_filling/4*, *draw_line/3* los cuales fueron creados usando la lógica y geometría del hexágono y apoyándonos en conocimientos aprendidos sobre la librería pce.

utils_visual: Aquí tenemos al predicado *check_positions_in_hand* para saber si dado un click este fue en una ficha que no está en la colmena; *check_position_in_hive* para saber si dado un click este fue en una ficha que está en la colmena; *flat_hex_corner* para encontrar todos los vértices de un hexágono; *new_image* crea una imagen en una posición; *write_message* escribe un mensaje en el tablero.

Para ejecutar el juego corremos el archivo game.pl .