Informe Azul

Alejandra Monzón C312
Marié del Valle C311

Dinámica del juego

El juego comienza en la regla `new_game(N)`, donde `N` son la cantidad de jugadores que participan en el juego.

En la simulación del juego se destacan dos hechos que almacenan los estados del juego y de los jugadores. Dichos hechos son `game_state(N,Bag,Factories, Center,Stack,InitialPlayer,CurrentPlayer,State)` y `player_state(Player, Wall,Pattern,Floor,Score)` respectivamente.

La simulación comienza inicializando la bolsa y las fábricas con los azulejos y los jugadores, con el muro, patrón y piso creados. Los azulejos que se ponen en las fábricas se eligen aleatoriamente y se eliminan de la bolsa. Además se elige aleatoriamente un jugador inicial, que comienza la `fase_1`.

El objetivo es, una vez iniciado el juego con `State = inicialized`, buscar si existe una secuencia de estados (`game_state`move) que permitan alcanzar el `State = win`, pasando por diferentes estados como `State = {fase_1, fase_2, fase_3, over}`.

```
311 > move(game_state(N,Bag,Factories,Center,Stack,InitialP,CurrentP,initialized),game_state(N,Bag,FFactories,FCenter,FStack,FInitialP,FCurrentP,fase1)):- ···
329 > move(game_state(N,Bag,[],[init_tile],Stack,InitialP,CurrentP,fase1),game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase2)):-···
332 > move(game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase1),game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase2)):-···
335 > move(game_state(N,Bag,Factories,Center,Stack,InitialP,CurrentP,fase1),game_state(N,Bag,FFactories,FCenter,FStack,FInitialP,FCurrentP,fase1)):- ···
349 > move(game_state(N,Bag,Factories,Center,Stack,InitialP,CurrentP,fase1),game_state(N,Bag,Factories,FCenter,FStack,FInitialP,FCurrentP,fase1)):- ···
362 > move(game_state(N,Bag,Factories,Center,Stack,InitialP,CurrentP,fase1),game_state(N,Bag,Factories,FCenter,FStack,FInitialP,FCurrentP,fase1)):- ···
375 > move(game_state(N,Bag,Factories,Center,Stack,InitialP,CurrentP,fase1),game_state(N,Bag,FFactories,FCenter,FStack,FInitialP,FCurrentP,fase1)):-···
387 > move(game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase2),game_state(N,Bag,[],[],Stack,InitialP,CurrentP,over)):-···
391 > move(game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase2),game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase3)):- ···
394 > move(game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase2),game_state(N,Bag,[],[],FStack,InitialP,FCurrentP,fase2)):-···
401 > move(game_state(N,[],[],[],[],InitialP,CurrentP,fase3),game_state(N,NBag,Factories,Center,Stack,InitialP,InitialP,over)):-···
404 > move(game_state(N,Bag,[],[],Stack,InitialP,CurrentP,fase3),game_state(N,NBag,Factories,Center,Stack,InitialP,InitialP,fase1)):-···
409 > move(game_state(N,[],[],[],Stack,InitialP,CurrentP,fase3),game_state(N,NBag,Factories,Center,[],InitialP,InitialP,fase1)):-···
414 > move(game_state(N,_,[],[],_,_,_,over),game_state(N,_,[],[],_,Winners,_,win)):- player_list(N,List), aditional_score(List,Scores), ···
```

Fase 1

En esta fase, el jugador inicial, empieza el juego, eligiendo robar de una fábrica; luego continúa el juego con el siguiente jugador, el cual se define como el hecho `next_player(CurrentPlayer,NumberOfPlayers,NextPlayer)`.

En el centro se encontrará la ficha inicial(`init_tile`), hasta que algún jugador decida robar azulejos del centro, designándose a sí mismo el próximo jugador inicial de la futura `fase_1`.

```
75   next_player(1,_,2).
76   next_player(2,2,1):-!.
77   next_player(2,N,3):-N>2.
78   next_player(3,3,1):-!.
79   next_player(3,N,4):-N>3.
80   next_player(4,4,1).
```

Método de elegir jugada

El método `best_move` consistiría en lo siguiente:

Cada jugador en dependencia del estado de su tablero (patrón, muro), tiene un conjunto de jugadas factibles según los azulejos disponibles en las fábricas y el centro. Estas jugadas se definen como válidas o factibles, si cumplen que para un color $\boldsymbol{x}$ de azulejos, en una fila $\boldsymbol{r}$ del patrón, no se ha puesto en la fila $\boldsymbol{r}$ del muro azulejo de color $\boldsymbol{x}$ y en la fila $\boldsymbol{r}$ del patrón no se ha puesto azulejo de color distinto de $\boldsymbol{x}$. Esto se verifica en la siguiente regla:

```
138   valid_move(Color,PatternRow,WallRow):-
139       not(member(Color, WallRow)),
140       not(full(PatternRow)),(empty(PatternRow); member(Color, PatternRow)).
```

A partir de dichas jugadas factibles, se seleccionan las mejores jugadas por fábricas y centro, atendiendo a un evaluación golosa que se le asigna a cada jugada. La siguente regla se define como `evaluate(Color,ZeroList,Floor,Evaluation, BonnusPoints)`.

```
161   evaluate(Color,ZeroList,Floor,Eval,Bonnus):- length(Color, C), length(ZeroList, Z) , E is C - Z, E > 0, not(full(Floor)), Eval is E * -3,!.
162   evaluate(Color,ZeroList,Floor,Z,Bonnus):- length(Color, C), length(ZeroList, Z) , E is C - Z, E=:=0, !.
163   evaluate(Color,ZeroList,Floor,Eval,Bonnus):- length(Color, C), length(ZeroList, Z) , E is C - Z, E > 0,full(Floor), Eval is Bonnus - E, !.
164   evaluate(Color,ZeroList,Floor,Eval,Bonus):- length(Color, C), length(ZeroList, Z) , E is C - Z, E < 0, Eval is Bonnus + E.
```

Esta regla se divide en varios casos:

1. La cantidad azulejos de color `Color`, es mayor que la cantidad de espacios disponibles en la fila del patrón y el piso no está lleno, por lo que se restarían puntos al jugador por los azulejos sobrantes que se pondrían

en el suelo. Por ello, se penaliza con -3 puntos por cada azulejos que sobre.

2. La cantidad de azulejos de color `Color`, es igual a la cantidad de espacios vaciós en la fila del patrón disponible, por lo que se asigna, como evaluación, la cantidad de espacios vacíos que se llenarán con los azulejos.

3. La cantidad de azulejos de color `Color`, es mayor que los espacios disponibles en la fila del patrón y el suelo está lleno, por tanto no se pierden puntos por los azulejos sobrantes y se completaría la fila. La evaluación de estas jugadas es igual a la cantidad de azulejos en la fila, menos la cantidad de azulejos que se descartarían.

4. La cantidad de azulejos de color `Color`, es menor que los espacios disponibles en la fila del patrón. En este caso se penalizaría por los espacios que quedan disponibles, luego de poner los azulejos.

Importante destacar que poner los azulejos directamente en el suelo es una jugada válida y que en caso de que el suelo este lleno, esos azulejos irían directo a la tapa de la caja.

Luego de elegir una mejor jugada por fábricas y centro, se selecciona la mejor de ellas y se ejecuta esta, actualizando el patrón y/o el suelo del jugador, así como la tapa de la caja, en algunos casos.

```
140  %Obtener jugada optima dentro de las factibles atendiendo al estado del jugador mediante algoritmo greedy
141  best_move([[Eval|BMove]],BMove,Eval):-!.
142  best_move([[Eval|BMove]|List],BestMove,Ev):- best_move(List,X,E), best(E,Eval,X,BMove,BestMove,Ev),!.
143
144
145  list_best_factory_move([],_,_,[]):-!.
146  list_best_factory_move([F|Fr],[FMoves|FrMoves],Floor,[[Eval|BMove]|Rest]):-
147      best_factory_move(F,FMoves,Floor,BMove,Eval), list_best_factory_move(Fr,FrMoves,Floor,Rest).
148
149      best_factory_move(Factory,[],Floor,[Factory,Color,PRow],-1000):-!.
150
151      best_factory_move(Factory,[[Color,floor]|RMoves],Floor,Move,Eval):-
152          split_for_item(Factory,Color,ColorList,_),split_for_item(Floor,0,ZeroList,_),
153          evaluate(ColorList,[],Floor,E1,0),best_factory_move(Factory,RMoves,Floor,Move2, E2),
154          best(E1,E2,[Factory,Color,floor],Move2,Move,Eval),!.
155
156      best_factory_move(Factory,[[Color,PatternRow]|RMoves],Floor,Move,Eval):-  PatternRow\=floor,
157         split_for_item(Factory,Color,ColorList,_),split_for_item(PatternRow,0,ZeroList,BonList), length(BonList,B),
158         evaluate(ColorList,ZeroList,Floor,E1,B), best_factory_move(Factory,RMoves,Floor,Move2, E2),
159         best(E1,E2,[Factory,Color,PatternRow],Move2,Move,Eval),!.
```

Condición de Parada

La `fase_1` finaliza cuando no queden azulejos disponibles en las fábricas y en el centro.

Fase 2

En esta fase, se actualiza el muro de los jugadores, poniendo en ellos los azulejos correspondientes a las filas llenas de los patrones y se actualiza la puntuación de los jugadores.

Método para poner azulejos en Muro

La acción de actualizar el patrón se define como una revisión de cada fila de el patrón en busca de una llena de azulejos, para trasladar un ejemplar de dicha fila hacia el muro y los restantes hacia la tapa de la caja.

Una vez puesto el azulejo en el muro, se realiza el conteo de los puntos que se deben adicionar al jugador.

La regla `cover_wall` realiza el procedimiento anteriormente descrito.

```
233  % REVESTIR MURO
234  cover_wall([],[],[],[],[],Stack,Stack,Transpose,[],0):-!.
235  cover_wall([P|Pattern],[W|Wall],[W|NWall],[0|Color],[P|NewP],Stack,NewStack,Transpose,[X|Xr],Score):-
236      not(full(P)),transpose(W,W1),multi_concat(Transpose,W1,UT1),
237     cover_wall(Pattern,Wall,NWall,Color,NewP,Stack,NewStack,UT1,Xr,Score),!.
238
239  cover_wall([[Col|P]|Pattern],[W|Wall],[NW|NWall],[Col|Y],[NP|NewP],Stack,NewStack,Transpose,[X|Xr],Score):-
240      full([Col|P]),
241      update_wall(Col,W,X,NW),
242      concat_list(Stack,P,Stack2),
243      replace_all(Col,0,[Col|P],NP),
244      transpose(NW,W1), multi_concat(Transpose,W1,UT1),
245      matrix_transpose(Wall,TW),multi_concat(UT1,TW,UT2),
246      nth0(Index,NW,Col),
247      nth0(Index,UT2,TWRow),
248      score_counter(NW,TWRow,Col,S1),
249      cover_wall(Pattern,Wall,NWall,Y,NewP,Stack2,NewStack,UT1,Xr,S2), Score is S1 + S2.
250
251
252  update_wall(_,[],[],[]):-!.
253  update_wall(Color,[_|Wr],[Color|Xr],[Color|Wr]):-!.
254  update_wall(Color,[W|Wr],[X|Xr],[W|Result]):- Color\=X, update_wall(Color,Wr,Xr,Result).
```

Cada vez que se traslada un azulejos de una fila del patrón a la fila correspondiente del muro, se verifica la adyacencia horizontal y vertical con otros azulejos para saber en cuanto incrementar la puntiación del jugador.

```
% PUNTUACION POR REVESTIR EL MURO

score_counter(Wall,TWall,Color,FScore):-
    horizontal_score(Wall,Color,HScore),
    horizontal_score(TWall,Color,VScore),
    add1(VScore,RVScore),
    FScore is HScore + RVScore + 1,!.

add1(0,0):-!.
add1(Score,NScore):- NScore is Score +1.

horizontal_score(Row,Color,Score):-
    adj(Row,Color,S),
    reverse(Row,Reverse),
    adj(Reverse,Color,S1),
    Score is S+ S1.

score_counter_floor(Floor,Score):- floor_values(F), scalar_product(Floor,F,Score).

adj([],Col,0):-!.
adj([Col|X],Col,Z):-score(X,Z),!.
adj([X|Y],Col,Z):- X\=Col, adj(Y,Col,Z).

score([],0):-!.
score([0|_],0):-!.
score([X|Y],Score):- score(Y,PartialScore), Score is PartialScore + 1.
```

Además, antes de actualizar el muro y puntuación del siguiente jugador, se restan los puntos correspondientes a los azulejos en el suelo del jugador y pasar estos a la tapa de la caja. Si entre dichos azulejos se encuentra la ficha inicial, pasar esta al centro.

Condición de parada

La `fase_2` termina en dos posibles casos:

1. Se termina de actualizar muro de todos los jugadores y ninguno completo una fila de dicho muro, por lo que se pasa a `fase_3`.

2. Al analizar a todos los jugadores, al menos uno tiene al menos una fila completa de azulejos, con lo cual se pasa al estado `over` del juego.

Fase 3

En la `fase_3`, se inicializan con azulejos de la bolsa las fábricas y se verifica que en el centro se encuentre la ficha inicial. En el caso que la bolsa esté vacía se pasan los azulejos de la tapa de la caja a la bolsa para proceder al llenado de las fábricas.

Condición de parada

Tiene dos condiciones de parada:

1. Si hay azulejos en bolsa o en la tapa de la caja se llenan cuantas fábricas sean posible y se pasa a `fase_1`

2. En caso contrario, no quedan más jugadas posibles a realizar por los jugadores, ya que no hay azulejos en las fábricas, por lo que se pasa al estado `over` del juego.

Estado Over

Aquí, como se termina el juego, se agrega a cada jugador los puntos adicionales por completar fila, columna o color de azulejo. Además se selecciona él(los) ganador(es), atendiendo a la mayor cantidad de puntos y en caso de empate a la mayor cantidad de filas completadas. De este estado se pasa directamente al estado `win`, el cual dará por terminada la simulación del juego.