Proyecto
Inteligencia Artificial

Garay, Iñaki (L.U. 67387) Montenegro, Emiliano (L.U. 83864)

Segundo Cuatrimestre 2010

Contents

Chapter 1

Archivos y Estructura del Agente

El proyecto entregado en esta etapa consta de 5 módulos principales:

1. `./agente_emignator.pl` : Predicados principales del agente.
2. `./auxiliary_predicates.pl` : Predicados auxiliares del agente.
3. `./update_state.pl` : Módulo de actualización del estado interno del agente.
4. `./decide_action.pl` : Módulo de toma de decisiones y razonamiento del agente.
5. `./extras_for_agents.pl` : Módulo entregado por catedra.

Todos los módulos se cargan cuando se ejecuta el agente pero se decidió dividir la funcionalidad para mantener la modularidad y trazabilidad del código y utilizar los predicados de manera transparente.

Chapter 2

Agente `emignator`

`agent_emignator.pl`

En este sector del agente, se agrupan los predicados que realizan la puesta en marcha del agente en el entorno. En primer lugar, se cargan los módulos adicionales del agente explicados en el gráfico y comienza a ejecutarse el ciclo del predicado run.

En este ciclo se realizan las siguientes acciones:

1. acciones de solicitud de percepción;
2. muestra de la percepción actual recibida;
3. se actualiza el estado interno del agente con la precepción tomada del entorno dada por el rango de visión del agente;
4. luego del actualizado del estado, se decide realizar un volcado por pantalla de la memoria actual del agente para reflejar que los datos tomados fueron realmente mapeados en su estado interno;
5. terminado esto y con el estado interno actualizado, se decide tomar la decisión de la acción a realizar en el siguiente turno. Esto hace que dependiendo de los datos adquiridos, se modifique potencialmente el objetivo que el agente tendrá y su acciones a realizar;
6. por ltimo, se efectiviza la acción mediante el predicado do_action/1 y se realiza la invocación nuevamente al predicado run para realizar un nuevo ciclo de ejecución del agente en el entorno.

Otros predicados que se pueden encontrar aquí son `start_ag/0` y `start_ag_instance/1`. Estos predicados incorporan al funcionamiento del agente porque hacen que este se conecte al entorno de combate dado por la cátedra. El segundo de los predicados, permite incorporar varias instancias de un mismo agente, dándole un identificador distintivo a cada uno. Por otro lado, el primer predicado sólo permite agregar un solo agente de esta clase.

Chapter 3

Representación del estado interno del agente

Experiencia previa de programación con Prolog ha demostrado que el uso de los meta-predicados `assert/1` y `retract/1` es una herramienta poderosa pero peligrosa, dificultando el testeo, eliminación de errores, y razonamiento acerca del código. Por esta razón se procuró minimzar el uso de predicados de alto nivel, teniendo solo dos predicados definidos como `:- dynamic`: `agent_state/6` y `agent_map/4`.

El modelo del mundo exterior que mantiene el agente se representa mediante el predicado `agent_map/4`. El primer argumento es la posición de la celda. El segundo argumento es el tipo de terreno de la celda. El tercer argumento es la lista de objetos avistados en esa celda. El cuarto argumento es el turno en el cual fue actualizado por última vez esta celda. Esto permite mantener un registro de la antigüedad de la información que tiene el agente.

El estado del agente se representa mediante el predicado dinámico `agent_state/6`.

El primer argumento mantiene la meta actual del agente. Este consta de un átomo que indica que objetivo inmediato desea lograr el agente. El agente puede estar explorando, en caso de que no conozca de ningun tesoro; viajando hacia un tesoro conocido para recolectarlo; evadiendo enemigos o huyendo hacia una posada, en caso que este malherido; atacando a otro agente o persiguiendolo, si la situación lo amerita.

Esto le provee al diseño flexibilidad a la hora de incorporar nuevos comportamientos.

El uso de la meta explícita, su actualización en base a las percepciones y la decision de la acción a tomar en base a ella aú no esta implementado.

El segundo argumento es el historial de percepciones. Mantiene una lista cuya cabeza es siempre la última percepcion recibida. El agente recuerda todas las percepciones y las utilizará para sacar conclusiones acerca de los otros agentes y sus acciones.

El tercer argumento es el historial de acciones tomadas por el agente. Mantiene una lista

cuya cabeza es siempre la útima acción tomada.

El cuarto argumento es la base de datos de los agentes conocidos. Mantiene el nombre del agente, la posición donde fue visto por ultima vez, si tiene tesoros y si el agente se ha clasificado como agresivo o no, en base a las acciones que se lo ha percibido tomar.

El quinto argumento es la lista de posadas conocidas. Es representada por una lista de posiciones.

El sexto argumento es la lista de tesoros conocidos sin recolectar. Es representada por una lista donde cada elemento es un lista con la posicion del tesoro y el turno en el cual fue avistado por ultima vez.

3.1 Proceso de actualización del estado interno

El principal predicado encargado de iniciar el proceso de actualización del estado del agente es `update_state/2`.

```
%-----------------------------------------------------------------------------------------------%
% update_state/2
% update_state(+Target, +Value)
%
% Update_state actualiza las varias partes de del estado interno del agente.
% El primer argumento es que parte actualizar.
% El segundo argumento es el valor con el cual actualizarlo.
% El primer argumento debe ser o bien el atomo current_goal, o bien el atomo percept.
% En el primer caso se cambia la meta actual del agente al valor del segundo argumento.
% En el segundo caso se actualiza todo el estado del agente segun la informacion provista en la
% percepcion, que esta ligada al segundo argumento.
%
% En el caso de una actualizacion del estado segun una percepcion nueva, el trabajo de la extraccion
% de la informacion la hace el predicado extract_information_from_vision/9

%-----------------------------------------------------------------------------------------------%
% extract_information_from_vision/9
% extract_information_from_vision(  +Turn,
%                                   +Vision,
%                                   +Agents_old,
%                                   +Inns_old,
%                                   +Treasures_old,
%                                   -Agents_new,
%                                   -Inns_new,
%                                   -Treasures_new,
%                                   -Last_Action)
%
% Dada la vision de una percepcion, el turno correspondiente a la percepcion, y las bases de datos
% del agente de agentes conocidos, posadas conocidas y tesoros conocidos, liga el sexto, septimo
% y octavo argumentos con las bases de datos de agentes, posadas y tesoros actualizadas segun la
% informacion presente en la percepcion, y liga el ultimo argumento a la ultima accion realizada
% por el agente.
%
% Por cada elemento de la vision, que consiste en la lista de tres elementos [Pos, Land, Things],
% primero actualiza el mapa del mundo del agente, luego elimina los tesoros conocidos que por
% alguna razon ya no se ven (las posadas no deben ser eliminadas porque no desaparecen) y por
% ultimo llama al predicado extract_information_from_cell/9 que realiza la extraccion de la
% informacion de esa celda en particular, segun sea el caso de que objeto se encuentra en la celda.
```

```
%-----------------------------------------------------------------------------------------------------%
% extract_information_from_cell/9
% extract_information_from_cell(Position
%                               Things_at_position,
%                               Agents_old, Inns_old, Treasures_old,
%                               Agents_new, Inns_new, Treasures_new,
%                               Last_action)
%
% Revisa los objetos encontrados en una celda, y calcula las nuevas listas del estado del agente.

%-----------------------------------------------------------------------------------------------------%
% update_agents/5
% update_agents(+Agents_old, -Agents_new, +Agent_Name, +Agent_Description, +Turn)
%
% Actualiza la base de datos de otros agentes.
% El primer argumento es la base de datos de agentes actual.
% El segundo argumento sera ligado a la base de datos actualizada con la informacion provista.
% El tercer argumento es el nombre del agente avistado.
% El cuarto argumento es la descripcion del agente avistado.
% El quinto argumento es el turno en que fue avistado por ultima vez el agente.

%-----------------------------------------------------------------------------------------------------%
% update_inns/3
% update_inns(+Inns_Old, -Inns_New, +Pos)
%
% Actualiza la lista de posadas conocidas.
% El primer argumento debe ser la lista actual de posadas conocidas por el agente.
% El segundo argumento sera unificado con la lista actualizada. Si la posada ya era conocida, sera
% la misma lista.
% El tercer argumento debe ser la posicion de la posada.

%-----------------------------------------------------------------------------------------------------%
% update_treasures/3
% update_treasures(+Treasures_old, -Treasures_new, +Pos)
%
% Actualiza la lista de tesoros avistados pero no recolectados por el agente.
% El primer argument debe ser la lista actual de tesoros conocidos por el agente.
% El segundo argumento sera unificado con la lista actualizada. Si el tesoro ya habia sido visto
% por el agente y se encuentra en la lista, entonces el resultado sera identico al primer argumento.
% El tercer argumento debe ser la posicion del tesoro avistado.

%-----------------------------------------------------------------------------------------------------%
% remove_missing_treasures/3
% remove_missing_treasures(+Treasures_Old, +Position, -Treasures_New)
%
% Si no hay un tesoro en la posicion, eliminarlo de la base de datos de tesoros.
% El primer argumento debe ser la lista de posiciones de los tesoros conocidos.
% El segundo argumento debe ser la posicion a eliminar.
% El tercer argumento es de salida, y es ligado a la lista actualizada.
```

Chapter 4

Toma de decisiones y razonamiento

El módulo `decide_action.pl` es el módulo en el cual se implementará la estrategia en la siguiente etapa. Por el momento, y de manera preliminar, se decidió incorporar la estrategia de movilidad dada por el `agent_barb.pl` pero con sutilezas de cambio que lo hacen más competitivo. La esencia del agente se basa en decidir acciones las cuales son los átomos de la estrategia a seguir. Para nuestro agente la estrategia seguida es la siguiente, estableciendo en los ítems la prioridad de las acciones segn se presentan y siendo los cuerpos del predicado `decide_action/1`. En el caso de que exista un agente en el rango de ataque, se toma la acción de atacar a ese agente de manera reactiva y sin realizar ningn tipo de análisis de la información almacenada hasta el momento. En la entrega siguiente se utilizará la información almacenada para determinar mejor la estrategia de ataque a otro agente. Por otro lado, para determinar si un agente está en una situación vulnerable de ataque, se recorren las 4 posibles celdas de ataque de la percepción, y si alguna de estas contiene un agente, se decide atacarlo.

Si en la celda en la que se encuentra el agente, también hay una posada, entonces el agente decide quedarse hasta llenar su stamina al máximo antes de salir. En el caso de que aparece un agente en la posición atacable, entonces decide atacarlo mientras recarga su energía.

Las siguientes versiones del predicado `decide_action/1` corresponden a decidir girar a la izquierda o derecha (la derecha o izquierda se determina mediante el punto cardinal al cual está mirando actualmente) dependiendo si en esas direcciones hay un tesoro o una posada.

Al fijarse en la percepción, si encuentra que entre los atributos de la celda que se encuentra a su derecha hay una posada o un tesoro, entonces toma como decisión girar a la derecha.

Análogamente al anterior, si entre los atributos de la celda de la izquierda, encuentra que existe un tesoro o una posada, decide girar a la izquierda como siguiente acción a realizar.

En el caso de que en la posición donde se encuentra el agente, también exista un tesoro, el agente realizará tomar como acción. Esto hará que el agente tome y guarde en su inventario este tesoro.

Si ninguna de las acciones anteriores puede realizarse, se toma como acción por defecto tomar un paso a la posición de enfrente a la cual el agente se encuentra mirando en ese momento.

4.1 Potenciales mejoras del agente

Las mejoras contempladas al comportamiento del agente para la segunda etapa incluyen:

- Identificación de situaciones de ataque por parte de otros agentes.
- Identificación de comportamiento agresivo de agentes sobre terceros.
- Decisión de la acción a tomar en base a la meta del agente.
- Razonamiento acerca de la stamina del agente. Cuando este casi por debajo del costo del camino mas corto a la posada mas cercana, ir hacia la posada.
- Estimación de la stamina de otros agentes basados en el historial de percepciones para decidir si combatir o huir.
- Estimación de los tesoros llevados por otros agentes, basado en el historial de percepciones.
- Identificación de agentes caminando en direccion a posadas como malheridos, como posible blanco a ser atacado.
- Identificación de situación de combate entre otros agentes, para mantenerse a la espera y atacar al ganador, probablemente debilitado.
- Identificación del área de ataque de otros agentes, para mantenerse fuera de ella.

Chapter 5

Predicados Auxiliares

Los predicados auxiliares se encuentran ubicados en `auxiliar_predicates.pl`. Este módulo se encarga de realizar la muestra por pantalla del estado interno del agente. En las impresiones se muestran los siguientes atributos:

Nombre del agente; Meta actual del agente, respecto a la estrategia a seguir en los próximos turnos; ltima acción realizada; Muestra la lista de agentes conocidos, con el historial de acciones realizadas mientras estuvo en el rango de vista; Muestra la lista de posadas que recuerda con sus posiciones; Muestra la lista de tesoros que vio y todavía faltan ser juntados;

Cada uno de los datos almacenados que posee una estructura particular, la cual se recorre debidamente para determinar los datos almacenados y luego imprimirlos por pantalla. Esta impresión se realiza adecuadamente para respetar el formato virtual, es decir, imprimir el correspondiente dato virtual con su semejante carácter o caracteres. Por otro lado, también se incorporó en este módulo un predicado que recorre la percepción recibida por el entorno y la muestra por pantalla. Esto sirve para comparar los datos almacenados, con los datos que maneja el entorno y chequear que la actualización de la memoria se hace correctamente.

Chapter 6

Código

6.1 agent_emignator.pl

```
:- consult('ag_primitives.pl').
:- consult('extras_for_agents.pl').
:- consult('auxiliary_predicates.prolog').
:- consult('update_state.prolog').
:- consult('decide_action.prolog').

:- dynamic ag_name/1.

%-----------------------------------------------------------------------------------------------%
s :- start_ag.

si(InstanceID) :- start_ag_instance(InstanceID).

start_ag :-
    AgName = emignator,
    register_me(AgName, Status),
    !,
    write('REGISTRATION STATUS: '), write(Status), nl, nl,
    Status = connected,
    assert(ag_name(AgName)),
    assert(agent_state(none, [none], [none], [], [], [])),
    run.

start_ag_instance(InstanceID) :-
    AgClassName = emignator,
    AgInstanceName =.. [AgClassName, InstanceID],
    register_me(AgInstanceName, Status),
    !,
    write('REGISTRATION STATUS: '), write(Status), nl, nl,
    Status = connected,
    assert(ag_name(AgInstanceName)),
    assert(agent_state(none, [none], [none], [], [], [])),
    run.

run :-
    get_percept(Percept),
    %-----------------------------------------------------------------------------------------%
    % AGENT CODE (internal state update and action choice)
    update_state(percept, Percept),
    display_agent_state,
    decide_action(Action),
    write('Action taken: '), write(Action), nl,
    %-----------------------------------------------------------------------------------------%
    do_action(Action),
    run.
```

6.2 update_state.pl

```
%-----------------------------------------------------------------------------------------------------%
% agent_state(?Current_Goal, ?Percept_History, ?Action_History, ?Agents, ?Inns, ?Treasures).
%
% Name:             nombre del agente, lo mismo que devuelve ag_name/1
% Current_Goal:     meta actual del agente
%        meta actual del agente, ya sea, evadir enemigos y buscar posada, o buscar tesoro,
%        explorar, atacar,
%            si es explorar (cuando no hay tesoro conocido) moverse a una posicion desconocida
%            mas cercana
%            si no hay posiciones sin explorar, ir a la posicion desconocida que se exploro hace
%            mas tiempo
%            si es juntar oro, ir hacia el oro mas cercano para juntar
%            si es atacar, ir al agente enemigo y atacar
%            si es huir, ir a la posada mas cercana evitando enemigos.
% Percept_History:  historial de percepciones, la primera es la ultima percepcion recibida
% Action_History:   historial de acciones, la primera es la ultima accion tomada
% Agents:           lista de agentes conocidos
%                       cada agente es [nombre del agente, posicion donde fue visto por ultima vez,
%                       ultima accion que se lo vio realizar, estimacion de su stamina,
%                       estimacion de su oro ]
%                       agentes agresivos: lista de nombres de agentes que lo han derrotado a
%                       otros agentes, incluyendose, por ende con mas probabilidad de ser agresivos
%                       o tener alta fight_skill, a ser tenidos en cuenta
%                       agentes debiles: lista de agentes
% Inns:             lista de las posiciones de las posadas conocidas
% Treasures:        lista de las posiciones de tesoros conocidos no recogidos
:- dynamic agent_state/6.

%-----------------------------------------------------------------------------------------------------%
% agent_map(Position, Land, Things, Turn_Discovered).
:- dynamic agent_map/4.

%-----------------------------------------------------------------------------------------------------%
% sample_percept(-Percept) unifica el argumento con una percepcion de prueba, para propositos de
% testeo.
sample_percept(Percept) :-
    Turn = 1,
    Vision = [
        [[06,06],    water, []],
        [[05,06],    water, []],
        [[04,06],    water, []],
        [[03,06],    plain, []],
        [[06,05],    plain, []],
        [[05,05],    plain, []],
        [[04,05],    water, []],
        [[03,05],    plain, []],
        [[06,07],    plain, [[agent,emig,[[dir,n],[unconscious,false],[previous_turn_action,move_fwd]]]]],
        [[05,07],    plain, [[treasure,t1,[[val,100]]]]],
        [[04,07],    plain, []],
        [[03,07],    plain, []],
        [[06,08],    plain, []],
        [[05,08],    plain, []],
        [[04,08],    plain, []],
        [[03,08],   forest, []],
        [[06,09], mountain, []],
        [[05,09],    plain, []],
        [[04,09],   forest, []],
        [[03,09],    plain, []]
        ],
    Attributes = [[pos, [2,3]], [dir, n], [stamina, 100], [max_stamina, 150], [fight_skill, 100]],
    Inventory = [],
    Percept = [Turn, Vision, Attributes, Inventory].

%-----------------------------------------------------------------------------------------------------%
% update_state/2
% update_state(+Target, +Value)
```

```
%
% Update_state actualiza las varias partes de del estado interno del agente.
% El primer argumento es que parte actualizar.
% El segundo argumento es el valor con el cual actualizarlo.
% El primer argumento debe ser o bien el atomo current_goal, o bien el atomo percept.
% En el primer caso se cambia la meta actual del agente al valor del segundo argumento.
% En el segundo caso se actualiza todo el estado del agente segun la informacion provista en la
% percepcion, que esta ligada al segundo argumento.
%
% En el caso de una actualizacion del estado segun una percepcion nueva, el trabajo de la extraccion
% de la informacion la hace el predicado extract_information_from_vision/9
update_state(current_goal, Value) :-
    agent_state(Current_Goal, Percept_History, Action_History, Agents, Inns, Treasures),
    replaceall(
        agent_state(Current_Goal, Percept_History, Action_History, Agents, Inns, Treasures),
        agent_state(Value,        Percept_History, Action_History, Agents, Inns, Treasures)
    ).

update_state(percept, [Turn, Vision, Attributes, Inventory]) :-
    agent_state(Current_Goal, Percept_History, Action_History, Agents_old, Inns_old, Treasures_old),
    %trace,
    extract_information_from_vision(Turn,
                                    Vision,
                                    Agents_old,
                                    Inns_old,
                                    Treasures_old,
                                    Agents_new,
                                    Inns_new,
                                    Treasures_new,
                                    Action),
    %notrace,
    replaceall(
        agent_state(Current_Goal,
                    Percept_History,
                    Action_History,
                    Agents_old,
                    Inns_old,
                    Treasures_old),
        agent_state(Current_Goal,
                    [[Turn, Vision, Attributes, Inventory] | Percept_History],
                    [Action | Action_History],
                    Agents_new,
                    Inns_new,
                    Treasures_new)
    ).

%-------------------------------------------------------------------------------------------------%
% Update State auxiliar predicates

%-------------------------------------------------------------------------------------------------%
% extract_information_from_vision/9
% extract_information_from_vision(  +Turn,
%                                   +Vision,
%                                   +Agents_old,
%                                   +Inns_old,
%                                   +Treasures_old,
%                                   -Agents_new,
%                                   -Inns_new,
%                                   -Treasures_new,
%                                   -Last_Action)
%
% Dada la vision de una percepcion, el turno correspondiente a la percepcion, y las bases de datos
% del agente de agentes conocidos, posadas conocidas y tesoros conocidos, liga el sexto, septimo
% y octavo argumentos con las bases de datos de agentes, posadas y tesoros actualizadas segun la
% informacion presente en la percepcion, y liga el ultimo argumento a la ultima accion realizada
% por el agente.
%
% Por cada elemento de la vision, que consiste en la lista de tres elementos [Pos, Land, Things],
% primero actualiza el mapa del mundo del agente, luego elimina los tesoros conocidos que por
```

```
% alguna razon ya no se ven (las posadas no deben ser eliminadas porque no desaparecen) y por
% ultimo llama al predicado extract_information_from_cell/9 que realiza la extraccion de la
% informacion de esa celda en particular, segun sea el caso de que objeto se encuentra en la celda.

% Si la Vision es una lista vacia, no hay nada que extraer.
extract_information_from_vision(_Turn,
                                [],
                                Agents1,
                                Inns1,
                                Treasures1,
                                Agents1,
                                Inns1,
                                Treasures1,
                                _Last_Action).

% Si hay elementos en la Vision, por cada uno extraer la informacion apropiada.
extract_information_from_vision(Turn,
                                [[Pos, Land, Things] | Vision],
                                Agents1,
                                Inns1,
                                Treasures1,
                                Agents3,
                                Inns3,
                                Treasures4,
                                Last_Action) :-
    replaceall(
        agent_map(Pos, _, _, _),
        agent_map(Pos, Land, Things, Turn)
    ),
    remove_missing_treasures(Treasures1, Pos, Treasures2),
    extract_information_from_cell(  Pos,
                                    Things,
                                    Agents1, Inns1, Treasures2,
                                    Agents2, Inns2, Treasures3, Last_Action),
    extract_information_from_vision(Turn,
                                    Vision,
                                    Agents2, Inns2, Treasures3,
                                    Agents3, Inns3, Treasures4, Last_Action).

%-----------------------------------------------------------------------------------------------------%
% extract_information_from_cell/9
% extract_information_from_cell(Position
%                                Things_at_position,
%                                Agents_old, Inns_old, Treasures_old,
%                                Agents_new, Inns_new, Treasures_new,
%                                Last_action)
%
% Revisa los objetos encontrados en una celda, y calcula las nuevas listas del estado del agente.

% Si la lista de objetos en la posicion es vacia, no hay nada que agregegar.
extract_information_from_cell(  _Pos,
                                [],
                                Agents1, Inns1, Treasures1,
                                Agents1, Inns1, Treasures1, _Last_Action).

% Hay un agente en la celda y soy yo, recuperar y devolver la ultima accion.
extract_information_from_cell(  Pos,
                                [[agent, ThingName, Description] | Things],
                                Agents1, Inns1, Treasures1,
                                Agents3, Inns3, Treasures3,  Last_Action) :-
    ag_name(My_Name),
    My_Name = ThingName,
    member([previous_turn_action, Last_Action], Description),
    extract_information_from_cell(  Pos,
                                    Things,
                                    Agents1, Inns1, Treasures1,
                                    Agents3, Inns3, Treasures3, Last_Action).

% Hay un agente en la celda y no soy yo, actualizar la lista de agentes.
```

```
extract_information_from_cell(  Pos,
                                [[agent, ThingName, Description] | Things],
                                Agents1, Inns1, Treasures1,
                                Agents3, Inns3, Treasures3, Last_Action) :-
    write('Encontre un AGENTE en la celda '), write_position(Pos), write(' ahora lo recordare.'), nl,
    update_agents(Agents1, Agents2, ThingName, Description, Turn),
    extract_information_from_cell(  Pos,
                                    Things,
                                    Agents2, Inns1, Treasures1,
                                    Agents3, Inns3, Treasures3, Last_Action).

% Hay una posada en la celda, actualizar la lista de posadas.
extract_information_from_cell(  Pos,
                                [[hostel, _ThingName, _Description] | Things],
                                Agents1, Inns1, Treasures1,
                                Agents3, Inns3, Treasures3, Last_Action) :-
    write('Encontre un HOTEL en la celda '), write_position(Pos), write(' ahora la recordare.'), nl,
    update_inns(Inns1, Inns2, Pos),
    extract_information_from_cell(  Pos,
                                    Things,
                                    Agents1, Inns2, Treasures1,
                                    Agents3, Inns3, Treasures3, Last_Action).

% Hay un tesoro en la celda, actualizar la lista de tesoros.
extract_information_from_cell(  Pos,
                                [[treasure, _ThingName, _Description] | Things],
                                Agents1, Inns1, Treasures1,
                                Agents3, Inns3, Treasures3, Last_Action) :-
    % Hay un tesoro en Pos, actualizar la lista de tesoros.
    write('Encontre un TESORO en la celda '), write_position(Pos), write(' ahora la recordare.'), nl,
    update_treasures(Treasures1, Treasures2, [Pos, Turn]),
    extract_information_from_cell(  Pos,
                                    Things,
                                    Agents1, Inns1, Treasures2,
                                    Agents3, Inns3, Treasures3, Last_Action).

%-----------------------------------------------------------------------------------------------%
% update_agents/5
% update_agents(+Agents_old, -Agents_new, +Agent_Name, +Agent_Description, +Turn)
%
% Actualiza la base de datos de otros agentes.
% El primer argumento es la base de datos de agentes actual.
% El segundo argumento sera ligado a la base de datos actualizada con la informacion provista.
% El tercer argumento es el nombre del agente avistado.
% El cuarto argumento es la descripcion del agente avistado.
% El quinto argumento es el turno en que fue avistado por ultima vez el agente.
update_agents(  [],
                [[Agent_Name,[Agent_Description], Turn]],
                Agent_Name,
                Agent_Description,
                Turn).

update_agents(  [[Agent_Name, Old_Agent_Description, _Old_Turn] | Tail],
                [[Agent_Name, Agent_New_Description, Turn] | Tail],
                Agent_Name,
                Agent_Description,
                Turn) :-
    append(Old_Agent_Description, [Agent_Description], Agent_New_Description).

update_agents(  [[Agent_Name_Foo | Resto] | Tail],
                [[Agent_Name_Foo | Resto] | Tail2],
                Agent_Name,
                Agent_Description,
                Turn) :-
    Agent_Name_Foo \= Agent_Name,
    update_agents(Tail, Tail2, Agent_Name, Agent_Description, Turn).

%-----------------------------------------------------------------------------------------------%
% update_inns/3
```

```
% update_inns(+Inns_Old, -Inns_New, +Pos)
%
% Actualiza la lista de posadas conocidas.
% El primer argumento debe ser la lista actual de posadas conocidas por el agente.
% El segundo argumento sera unificado con la lista actualizada. Si la posada ya era conocida, sera
% la misma lista.
% El tercer argumento debe ser la posicion de la posada.
update_inns(Inns_old, Inns_new, Pos) :-
    ord_add_element(Inns_old, Pos, Inns_new).

%-----------------------------------------------------------------------------------------------------%
% update_treasures/3
% update_treasures(+Treasures_old, -Treasures_new, +Pos)
%
% Actualiza la lista de tesoros avistados pero no recolectados por el agente.
% El primer argument debe ser la lista actual de tesoros conocidos por el agente.
% El segundo argumento sera unificado con la lista actualizada. Si el tesoro ya habia sido visto
% por el agente y se encuentra en la lista, entonces el resultado sera identico al primer argumento.
% El tercer argumento debe ser la posicion del tesoro avistado.
update_treasures([], [[[F, C], Turn]], [[F, C], Turn]).

update_treasures([[[F, C],Old_Turn] | Tail], [[[F, C], Turn] | Tail], [[F, C], Turn]) :-
    write('ReEncontre el TESORO de '),
    write('['), write(F), write(','), write(C), write(']'),
    write(' ahora lo recordare que sigue alli.'), nl.

update_treasures([[Pos1, Turn_Pos1] | Tail], [[Pos1, Turn_Pos1] | Treasures_new], [Pos2, Turn]) :-
    Pos1 \= Pos2,
    update_treasures(Tail, Treasures_new, [Pos2,Turn]).

%-----------------------------------------------------------------------------------------------------%
% agent_last_percept/1
% agent_last_percept(-Percept)
%
% Unifica el argumento con la ultima percepcion recibida por el agente.
agent_last_percept(Percept) :-
    agent_state(_, [Percept | _], _, _, _, _).

%-----------------------------------------------------------------------------------------------------%
% agent_last_action/1
% agent_last_action(-Action)
%
% Unifica el argumento con la ultima accion realizada por el agente.
agent_last_action(Action) :-
    agent_state(_, _, [Action | _], _, _, _).

%-----------------------------------------------------------------------------------------------------%
% agent_position/1
% agent_position(-Position)
%
% Unifica el argumento con la posicion actual del agente.
agent_position(Position) :-
    agent_state(_, [[_, _, Attributes, _] | _], _, _, _, _),
    member([pos, Position], Attributes).

%-----------------------------------------------------------------------------------------------------%
% agent_direction/1
% agent_direction(-Direction)
%
% Unifica el argumento con la direccion actual del agente.
agent_direction(Direction) :-
    agent_state(_, [[_, _, Attributes, _] | _], _, _, _, _),
    member([dir, Direction], Attributes).

%-----------------------------------------------------------------------------------------------------%
% agent_stamin/1
% agent_stamina(-Stamina)
%
% Unifica el argumento con el valor actual de stamina del agente.
```

```
agent_stamina(Stamina) :-
    agent_state(_, [[_, _, Attributes, _] | _], _, _, _, _),
    member([stamina, Stamina], Attributes).

%-----------------------------------------------------------------------------------------------------%
% agent_max_stamina/1
% agent_max_stamina(-Max_Stamina)
%
% Unifica el argumento con el valor maximo de stamina del agente.
agent_max_stamina(Max_Stamina) :-
    agent_state(_, [[_, _, Attributes, _] | _], _, _, _, _),
    member([max_stamina, Max_Stamina], Attributes).

%-----------------------------------------------------------------------------------------------------%
% agent_fight_skill/1
% agent_fight_skill(-Fight_Skill)
%
% Unifica el argumento con la habilidad de combate del agente.
agent_fight_skill(Fight_Skill) :-
    agent_state(_, [[_, _, Attributes, _] | _], _, _, _, _),
    member([fight_skill, Fight_Skill], Attributes).

%-----------------------------------------------------------------------------------------------------%
% remove_missing_treasures/3
% remove_missing_treasures(+Treasures_Old, +Position, -Treasures_New)
%
% Si no hay un tesoro en la posicion, eliminarlo de la base de datos de tesoros.
% El primer argumento debe ser la lista de posiciones de los tesoros conocidos.
% El segundo argumento debe ser la posicion a eliminar.
% El tercer argumento es de salida, y es ligado a la lista actualizada.
remove_missing_treasures([], _Pos, []).

remove_missing_treasures([[Pos,T] | Tail], Pos, Tail).

remove_missing_treasures([[Pos_Foo, Turn_Foo] | Tail], Pos, [[Pos_Foo, Turn_Foo] | New_Tail]) :-
    Pos_Foo \= Pos,
    remove_missing_treasures(Tail,Pos,New_Tail).
```

6.3 decide_action.pl

```
costados(n,w,e).
costados(s,e,w).
costados(e,n,s).
costados(w,s,n).

%-----------------------------------------------------------------------------------------------------%
decide_action(attack(Victim)) :-
    attackable_agent(Victim).

decide_action(Action) :-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    agent_position(Position),
    member([pos, Pos], Attrs),
    Position = Pos,
    member([Pos, _Land, Content], Vision),
    member([hostel, TrName, _], Content), agent_stamina(Stamina),
    agent_max_stamina(Max_Stamina),
    Stamina < Max_Stamina,
    write('Esperar en esta posada hasta recargar mi stamina.'), nl,
    Action = turn(none).

decide_action(Action) :-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    member([pos, Pos], Attrs),
    member([dir, Dir], Attrs),
    costados(Dir, Derecha, _Izquierda),
```

```
    ady_at_cardinal(Pos, Derecha, PosInRight),
    member([PosInRight, _Land, Content], Vision),
    (
      member([treasure, TrName, _], Content), write('Vi un TESORO a mi DERECHA y voy a juntarlo.'), nl
    ;
      member([hostel, TrName, _], Content),
      agent_stamina(Stamina), Stamina < 50,
      write('Voy a recargar mi salud en el HOTEL de mi DERECHA.'), nl
    ),
    Action = turn(Derecha).

decide_action(Action) :-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    member([pos, Pos], Attrs),
    member([dir, Dir], Attrs),
    costados(Dir, _Derecha, Izquierda),
    ady_at_cardinal(Pos, Izquierda, PosInLeft),
    member([PosInLeft, _Land, Content], Vision),
    (
      member([treasure, TrName, _], Content), write('Vi un TESORO a mi IZQUIERDA y voy a juntarlo.'), nl
    ;
      member([hostel, TrName, _], Content), agent_stamina(Stamina),
      Stamina < 50, write('Voy a recargar mi salud en el HOTEL de mi IZQUIERDA.'), nl
    ),
    Action = turn(Izquierda).

decide_action(Action) :-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    member([pos, Pos], Attrs),
    member([dir, Dir], Attrs),
    ady_at_cardinal(Pos, Dir, PosInFront),
    member([PosInFront, Land, _Content], Vision),
    (
      Land = water, write('Uh, me tropece con AGUA, voy a girar.'), nl
    ;
      Land = forest, write('Uh, me tropece con un BOSQUE, voy a girar.'), nl
    ),
    next_90_clockwise(DesiredDir, Dir),
    Action = turn(DesiredDir).


decide_action(Action) :-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    member([pos, MyPos], Attrs),
    member([MyPos, _Land, Content], Vision),
    member([treasure, TrName, _], Content),
    write('He encontrado un tesoro conocido como '), write(TrName), write('.'), nl,
    write('voy a intentar recogerlo...'),nl,
    Action = pickup(TrName).

decide_action(move_fwd).

%-----------------------------------------------------------------------------------------------%
attackable_agent(Victim):-
    agent_last_percept([_Turn, Vision, Attrs, _Inv]),
    member([pos, MyPos], Attrs),
    member([dir, MyDir], Attrs),
    pos_in_attack_range(MyPos, MyDir, PosInAttackRange),
    member([PosInAttackRange, _Land, Content], Vision),
    member([agent, Victim, AgProperties], Content),
    ag_name(MyName), %Debera llamarlo my_name!!!
    Victim \= MyName,
    member([unconscious, false], AgProperties).

%-----------------------------------------------------------------------------------------------%
%pos_in_attack_range(+MyPos, +MyDir, -PosInAttackRange)

pos_in_attack_range(MyPos, _MyDir, MyPos).
pos_in_attack_range(MyPos, MyDir, FrontPos) :-
```

```
    ady_at_cardinal(MyPos, MyDir, FrontPos).
pos_in_attack_range(MyPos, MyDir, FrontRightPos) :-
    ady_at_cardinal(MyPos, MyDir, FrontPos),
    next_90_clockwise(MyDir, NextDir),
    ady_at_cardinal(FrontPos, NextDir, FrontRightPos).
pos_in_attack_range(MyPos, MyDir, FrontLeftPos) :-
    ady_at_cardinal(MyPos, MyDir, FrontPos),
    next_90_clockwise(PrevDir, MyDir),
    ady_at_cardinal(FrontPos, PrevDir, FrontLeftPos).
```

6.4 auxiliary_predicates.pl

```
%-----------------------------------------------------------------------------------------------------%
display_agent_state :-
    ag_name(Name),
    agent_state(Current_Goal,
                [[Turn, Vision, Attributes, Inventory] | _Percept_History],
                [Last_Action | _Action_History],
                _Agents,
                Inns,
                Treasures),
    write('#################################################################################'), nl,
    write('Agent name:      '), write(Name), nl,
    write('Current goal:    '), write(Current_Goal), nl,
    write('Last action:     '), write(Last_Action), nl,
    write('Known Agents:    '), write_agents(Agents), nl,
    write('Inns:            '), write_inns(Inns), nl,
    write('Treasures  :     '), write_treasures(Treasures), nl,
    write('Inventory:      '), write_inventory(Inventory), nl, nl,
    write('Map :           '), nl, nl, write_all_map, nl,
    nl.

write_percept([Turn, Vision, Attributes, Inventory]) :-
    write('Percept:         '), nl,
    write('    Turn:        '), write(Turn), nl,
    write('    Vision:      '), nl,
    write_vision(Vision),
    write('    Attributes:  '), write_attributes(Attributes), nl,
    write('    Inventory:   '), write_inventory(Inventory), nl.

%-----------------------------------------------------------------------------------------------------%
write_vision([]).
write_vision([[Pos, Land, Things] | Vision]) :-
    write('        '), write_position(Pos), write(', '), write_land(Land), write(', '), write(Things), nl,
    write_vision(Vision).

%-----------------------------------------------------------------------------------------------------%
write_attributes(Attributes) :-
    write('['),
    write_attributes1(Attributes),
    write(']').

write_attributes1([]).
write_attributes1([Attribute]) :-
    write_attribute(Attribute).
write_attributes1([Attribute | Attributes]) :-
    write_attribute(Attribute),
    write(', '),
    write_attributes1(Attributes).

write_attribute([Attribute, Value]) :-
    write('['), write(Attribute), write(','), write(Value), write(']').

%-----------------------------------------------------------------------------------------------------%
write_inventory(Inventory) :-
```

```
    write(Inventory).

%-------------------------------------------------------------------------------------------------%
write_agents([]).
write_agents([[Agent_Name, _Resto, Turn] | TailAgents]) :-
    write('[Nombre agente :'), write(Agent_Name),
    write(', Ultimo turno visto: '), write(Turn), write('] '),
    write_agents(TailAgents).
    % Por comodidad no se imprime la informacin guardada debido a que se encuentra en estado
    % natural de la percepcion.

%-------------------------------------------------------------------------------------------------%
write_inns(Inns) :- write_position_list(Inns).

%-------------------------------------------------------------------------------------------------%
write_treasures([]).
write_treasures([[[X,Y],T]|Tail]) :-
    write(' ['),
    write('['),
    write(X),
    write(','),
    write(Y),
    write(']'),
    write(','),
    write(T),
    write('] '),
    write_treasures(Tail).

%-------------------------------------------------------------------------------------------------%
write_position_list(Positions) :-
    write('['),
    write_position_list1(Positions),
    write(']').

write_position_list1([]).
write_position_list1([Position]) :-
    write_position(Position).
write_position_list1([Position | Positions]) :-
    write_position(Position),
    write(', '),
    write_position_list1(Positions).

%-------------------------------------------------------------------------------------------------%
write_land(mountain) :- write('mountain').
write_land(water)    :- write('   water').
write_land(forest)   :- write('  forest').
write_land(plain)    :- write('   plain').

%-------------------------------------------------------------------------------------------------%
write_position(Position) :-
    position_to_string(Position, String),
    write(String).

position_to_string( [F, C], String ) :-
    number_to_string(F, Fs),
    number_to_string(C, Cs),
    string_concat( '[', Fs,   S1 ),
    string_concat( S1,  ',', S2 ),
    string_concat( S2,  Cs,   S3 ),
    string_concat( S3,  ']', String).

number_to_string( N, Ns ) :-
    N < 10,
    string_concat( '0', N, Ns ).
number_to_string( N, Ns ) :-
    string_concat( N, '', Ns ).

string_concatenation([], '').
string_concatenation([S], S1) :-
```

```
    string_concat(S, '', S1).
string_concatenation([S1, S2|T], S3) :-
    string_concat(S1, S2, Temp),
    string_concatenation([Temp|T], S3).

%-----------------------------------------------------------------------------------------------------%
write_all_map :-
    forall(agent_map(Pos, plain, _Things, Turn),
        (
            write(' ['),
            write_position(Pos),
            write(','),
            write_land(plain),
            write(', '),
            write(Turn),
            write('] ')
        )), nl, nl,
    forall(agent_map(Pos, water, _Things, Turn),
        (
            write(' ['),
            write_position(Pos),
            write(','),
            write_land(water),
            write(', '),
            write(Turn),
            write('] ')
        )), nl, nl,
    forall(agent_map(Pos, forest, _Things, Turn),
        (
            write(' ['),
            write_position(Pos),
            write(','),
            write_land(forest),
            write(', '),
            write(Turn),
            write('] ')
        )), nl, nl,
    forall(agent_map(Pos, mountain, _Things, Turn),
        (
            write(' [') ,
            write_position(Pos),
            write(','),
            write_land(mountain),
            write(', '),
            write(Turn),
            write('] ')
        )), nl.

%-----------------------------------------------------------------------------------------------------%
replace(X, Y) :- retract(X), !, assert(Y).

replaceall(X, Y) :- retractall(X), !, assert(Y).
```