# Proyecto
# Inteligencia Artificial

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

Segundo Cuatrimestre 2010

# Índice general

# Capítulo 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. `./update_state.prolog` : Módulo de actualización del estado interno del agente.
3. `./decide_action.prolog` : Módulo de toma de decisiones y razonamiento del agente.
4. `./searchprolog` : Módulo de busqueda de caminos y planes.
5. `./auxiliary_predicates.prolog` : Predicados auxiliares del agente.
6. `./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.

# Capítulo 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 último, 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.

# Capítulo 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/9` 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 intención actual del agente. Este consta de un átomo o estructura 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. Las diferentes intenciones que puede tener el agente y el proceso de determinacion de la acción a tomar en base a ésta se detalla mas adelante.

Las posibles intenciones son:

1. `none`
2. `gather_treasure`
3. `resting`
4. `go_to_hostel`

5. `defend_myself_from(Agent)`
6. `hunting(Agent)`
7. `attack(Agent)`
8. `explore`

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. Esta es representada por un arreglo asociativo en el cual las claves son los nombres de los agentes. El valor asociado a la clave es a su vez otro arreglo asociativo, cuyos valores almacenan la siguiente informacion segun la clave:

- `wealth` puede ser `rich` o `poor`, indica si el agente cree que el agente enemigo posee un tesoro en su inventario o no. Cuando el agente percibe a un agente enemigo por primera vez, se asume que es pobre. Si lo ve realizar la accion `pickup`, entonces asume que es rico. Si lo ve inconsciente, asume que perdio sus tesoros y es pobre.
- `dir` guarda la ultima direccion en la cual estaba mirando el agente enemigo.
- `pos` guarda la ultima posicion en la cual se lo percibio al agente enemigo.
- `action_history` mantiene una lista de las acciones que se lo vio realizar al agente enemigo.
- `attacked_by` mantiene el nombre de otro agente que ataco al agente enemigo.
- `harmed_by` mantiene el nombre de otro agente que lastimo al agente enemigo.

El quinto argumento es la lista de posadas conocidas. Este es representado por un arreglo asociativo en el cual las claves son las posiciones de las posadas. Los valores asociados a las claves consisten unicamente del turno en el cual se percibio a la posada, pero el objetivo era almacenar el ultimo turno en el cual se estuvo en la posada. De esta manera se puede determinar si es demasiado temprano para volver a entrar.

El sexto argumento es el conjunto de tesoros conocidos sin recolectar. Este es representado por un arreglo asociativo, en el cual las claves son el nombre del tesoro y los valores la información asociada a éste. Esta informacion incluye la posicion donde se vio al tesoro, la descripcion del tesoro recibida en la percepcion y el turno en el cual se lo vio.

El séptimo argumento almacena un camino que el agente ha calculado, representado como una lista de posiciones.

El octavo argumento almacena la secuencia de acciones necesarias para seguir el camino calculado, representada como una lista de acciones.

El noveno argumento es el costo del camino almacenado.

## 3.1. Predicados auxiliares de acceso al estado del agente

.

Para proveer una interfaz comoda al estado del agente, se utiliza un conjunto de predicados auxiliares

```
%-----------------------------------------------------------------------------------------------%
% agent_intention/1
% agent_intention(-Intention)
%
% Unifica el argumento con la intencion actual del agente.

\section{Proceso de actualizaci\'{o}n del estado interno}

El principal predicado encargado de iniciar el proceso de actualizaci\'{o}n del estado del agente
es \texttt{update\_state/2}.

%-----------------------------------------------------------------------------------------------%
% agent_last_percept/1
% agent_last_percept(-Percept)
%
% Unifica el argumento con la ultima percepcion recibida por el agente.

%-----------------------------------------------------------------------------------------------%
% agent_last_action/1
% agent_last_action(-Action)
%
% Unifica el argumento con la ultima accion realizada por el agente.

%-----------------------------------------------------------------------------------------------%
% agent_known_agents/1
% agent_known_agents(-Agents)
%
% Unifica el argumento con la base de datos de agentes enemigos.

%-----------------------------------------------------------------------------------------------%
% agent_known_hostels/1
% agent_known_hostels(-Hostels)
%
% Unifica el argumento con la base de hostels conocidos por el agente.

%-----------------------------------------------------------------------------------------------%
% agent_known_treasures/1
% agent_known_treasures(Treasures)
%
% Unifica el argumento con la base de datos de tesoros conocidos sin recolectar.

%-----------------------------------------------------------------------------------------------%
% agent_current_path/1
% agent_current_path(-Path)
%
% Unifica el argumento con el camino almacenado.
```

```
%-------------------------------------------------------------------------------------------------%
% agent_current_plan/1
% agent_current_plan(-Plan)
%
% Unifica el argumento con la secuencia de acciones almacenada.

%-------------------------------------------------------------------------------------------------%
% agent_current_path_cost/1
% agent_current_path_cost(-Path_Cost)
%
% Unifica el argumento con el costo del camino almacenado.

%-------------------------------------------------------------------------------------------------%
% agent_current_turn/1
% agent_current_turn(-Turn)
%
% Unifica el argumento con el turno actual.

%-------------------------------------------------------------------------------------------------%
% agent_position/1
% agent_position(-Position)
%
% Unifica el argumento con la posicion actual del agente.

%-------------------------------------------------------------------------------------------------%
% agent_direction/1
% agent_direction(-Direction)
%
% Unifica el argumento con la direccion actual del agente.

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

%-------------------------------------------------------------------------------------------------%
% agent_max_stamina/1
% agent_max_stamina(-Max_Stamina)
%
% Unifica el argumento con el valor maximo de stamina del agente.

%-------------------------------------------------------------------------------------------------%
% agent_fight_skill/1
% agent_fight_skill(-Fight_Skill)
%
% Unifica el argumento con la habilidad de combate del agente.
```

## 3.2. Predicados de actualizacion del estado del agente

```
%-------------------------------------------------------------------------------------------------%
% 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.
%
% Posibles valores para Target son:
%   current_intention: actualiza la intencion del agente
%   percept: actualiza todo el estado interno del agente con la informacion provista en la
%            percepcion y las inferencias que puede realizar a partir de ella.
%   current_path: actualiza el camino almacenado por el agente.
%   current_plan: actualiza la secuencia de acciones almacenada por el agente.
%   current_path_cost: actualiza el costo del camino almacenado por el agente.
%
```

```
% 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.
```

# Capítulo 4

# Toma de decisiones y razonamiento

El predicado `decide_action/1` que determina la accion que ejecutara el agente trabaja en dos fases: primero analiza la situacion actual segun las actualizaciones realizadas a partir de la percepcion recibida para determinar la intencion del agente este turno (en el predicado `determine_intention/0`). Una vez que se determino la intencion, la accion se determina a partir de esta (en el predicado `determine_action/1`).

Las intenciones tienen un orden de prioridad impuesto por el orden de las clausulas del predicado `determine_intention/1`.

## 4.1. Determinación de la intención del agente

El predicado `determine_intention/1` contempla las siguientes situaciones:

- **Situacion (1):** Si hay un tesoro en la posicion actual, su intencion sera juntar el tesoro.
- **Situacion (2):** Si no se da la situacion 1 y si el agente esta en un hostel y su stamina actual es menor a su maxima stamina, su intencion sera quedarse descansando hasta que lo echen.
- **Situacion (3):** Si no se dan las situaciones 1 o 2, y su stamina es menor o igual el costo del camino al hostel mas cercano mas un 10 % del costo, su intencion sera ir al hostel. Si no hay hostels, esto falla siempre. Explorara hasta que quede inconsciente por cansancio.
- **Situacion (4):** Si no se dan las situaciones 1, 2, o 3 y el agente fue atacado el turno anterior por otro agente, su intencion sera defenderse del agente enemigo. Como dijo Napoleon Bonaparte, "La mejor defensa es una buena ofensa."
- **Situacion (5):** Si no se dan las situaciones, 1, 2, 3 o 4 y la intencion anterior del agente era defenderse de un agente atacante o cazar a un agente atacante, entonces su intencion sera cazar al agente atacante. Cuando la intencion del agente es cazar a otro, si el agente esta en rango de ataque lo ataca, y sino se mueve hacia el agente que esta huyendo. El agente saldra del modo caceria cuando se den algunas de las situaciones 1, 2, 3 o 4 o cuando el agente atacante este inconsciente.

- **Situacion (6):** Si no se dan las situaciones 1, 2, 3, 4, o 5 y el agente estaba en modo defensa o caceria y el agente atacante esta inconsciente, y hay tesoros sin juntar, la intencion del agente sera juntar el tesoro mas cercano.
- **Situacion (7):** Si no se dan las situaciones 1, 2, 3, 4, 5 o 6 y hay tesoros conocidos por recolectar, entonces la intencion del agente sera ir al tesoro mas cercano.
- **Situacion (8):** Si no se dan las situaciones 1, 2, 3, 4, 5, 6 o 7 y hay un agente enemigo que el agente ha marcado como rico", entonces la intencion del agente sera atacar al agente enemigo rico.
- **Situacion (9):** Si no se da ninguna de las situaciones anteriores, el agente no tiene nada urgente que hacer y su intencion sera explorar.

## 4.2. Determinación de la acción dada la intención

El predicado correspondiente a la segunda fase de razonamiento determina la accion a partir de la intencion, y consiste de un conjunto que clausulas que se corresponden con las situaciones detalladas anteriormente.

- **Situacion (1):** Si la intencion del agent es descansar, no toma ninguna accion.
- **Situacion (2):** Si la intencion del agente es juntar un tesoro, recupera el nombre del tesoro en su posicion actual y realiza la accion pickup del tesoro correspondiente.
- **Situacion (3):** Si la intencion del agente es ir al hostel mas cercano para recuperar stamina, entonces recupera la secuencia de acciones que tiene que tomar para llegar al hostel, almacena el resto de las acciones y ejecuta la primera.
- **Situacion (4):** Si la intencion del agente es defenderse de otro agente atacante, y el agente esta en rango de ataque, entonces realiza la accion attack con el nombre de agente correspondiente. Se sabe que el agente atacante siempre estara en rango de ataque porque para que el agente haya sido atacado, el otro tiene que haber realizado el turno anterior la accion attack, y por ende esta en rango.
- **Situacion (5.1):** Si la intencion del agente es cazar a otro agente, y el agente atacante esta en rango, entonces realiza la accion attack con el nombre de agente correspondiente.
- **Situacion (5.2):** Si la intencion del agente es cazar a otro agente, y no esta en rango, entonces el agente calcula el camino mas corto hacia el agente y se mueve por ese camino. El camino se recomputa nuevamente cada vez para tomar en cuenta las acciones de evasion del agente enemigo.
- **Situacion (6) y (7):** Si la intencion del agente es ir a recolectar un tesoro que tenia pendiente, entonces el agente recupera el camino hacia el tesoro y lo sigue, y actualiza el plan actual con el resto de las acciones necesarias para llegar al tesoro.

- **Situacion (8.1):** Si la intencion del agente es atacar a otro agente, y el agente esta en rango, entonces el agente realizara la accion attack con el nombre de agente correspondiente.
- **Situacion (8.2):** Si la intencion del agente es atacar a otro agente y el agente no esta en rango de ataque pero si en rango visual, entonces el agente seguira el camino mas corto al agente enemigo. El camino se recomputa cada turno para tomar en cuenta las acciones evasivas del otro agente.
- **Situacion (9):** Si la intencion del agente es explorar, entonces seguira el camino hacia la celda no explorada que hace mas tiempo no se vio.

## 4.3. Implementación de la búsqueda

Los predicados que implementan la busqueda se encuentran en el modulo `search.prolog`.

### 4.3.1. Principales predicados

```
%-------------------------------------------------------------------------------------------------%
% search/6
% search(+Start_Direction, +Start_Position, +End_Position, -Path, -Actions,  -Cost)
%
% El 1er argument es la direccion en la cual esta mirando el agente al comenzar la busqueda.
% El 2do argumento es la posicion desde la cual se comienza la busqueda.
% El 3er argumento es la posicion a la cual se quiere llegar.
% El 4to argumento queda ligado al camino solucion desde la posicion inicial a la meta.
% El 5to argumento queda ligado a la secuencia de acciones necesaria para seguir el camino.
% El 6to argumento queda ligado al costo del camino.
%
% Si no hay camino, falla.

%-------------------------------------------------------------------------------------------------%
% multiple_search/4
% multiple_search(+Direction, +Position, +Destinations, -Paths)
%
% El 1er argumento es la direccion en la cual el agente esta mirando al comenzar la busqueda.
% El 2do argumento es la posicion desde la cual se comienza la busqueda.
% El 3er argumento es la lista de posiciones hacia las cuales se quieren calcular los caminos.
% El 4to argumento queda ligado a una lista de listas, donde cada elemento es el resultado de buscar
% un camino a uno de los destinos dados, respectivamente.
% Cada solucion es una lista de tres elementos, conteniendo el camino hacia el destino, la secuencia
% de acciones necesarias para llegar a el representada como una lista, y el costo de ejecutar esas
% acciones.
%
% multiple_search/4 nunca falla, si no hay un camino a ninguna posicion dada, devuelve la lista vacia.
```

### 4.3.2. Estructura de nodo del espacio de busqueda

```
%-------------------------------------------------------------------------------------------------%
% node(Position, Path_to_Node, Action_List, Path_Cost, Last_Facing_Direction)
%
% La estructura node/5 mantiene un nodo del grafo de busqueda.
% El 1er argumento es la posicion del mapa (el "estado" del nodo del grafo).
% El 2do argumento es el camino hacia la posicion desde donde se comenzo la busqueda, representado
% por una lista de posiciones.
% El 3er argumento es la lista de acciones que el agente debe ejecutar desde la posicion donde se
% comenzo la busqueda para llegar a el.
% El 4to argumento es el costo del camino, tomando en cuenta todas las acciones necesarias y el
```

```
% terreno del camino recorrido.
% El 5to argumento es la ultima direccion en la cual queda mirando el agente, mantenida para poder
% calcular las acciones apropiadas para la lista de acciones.
```

### 4.3.3. Heuristica utilizada

La heuristica utilizada es la distancia manhattan, calculada mediante el predicado `heuristic/3`.

### 4.3.4. Control de visitados

La generacion de vecinos se realiza mediante el predicado `neighbours/2`. Una vez generados los vecinos, se los procesa agregandoles el costo y la accion necesaria para llegar a el a sus estructuras, mediante el predicado `add_paths/3`. Una vez procesados los vecinos, el predicado `checkall/5` es llamado por `astar_add_to_frontier/5` y realiza el control de visitados y los agrega a la frontera. Posteriormente la frontera es reordenada por el valor f de los nodos.

```
%-----------------------------------------------------------------------------------------------%
% checkall/5
% checkall(Neighbours, Old_Frontier, New_Frontier, Old_Visited, New_Visited)
%
% Realiza un chequeo por cada vecino, y segun los resultados actualiza la Frontera y los Visitados.
%
% El 1er argumento es la lista de vecino a verificar.
% El 2do argumento es la frontera vieja.
% El 3er argumento es la frontera actualizada.
% El 4to argumento es la lista de nodos visitados vieja.
% El 5to argumento es la lista de visitados actualizada.
```

La verificacion realizada por `checkall/5` esta implementada en el predicado `check/` el cual realiza la verificacion para un nodo en particular.

```
%-----------------------------------------------------------------------------------------------%
% check/5
% check(+Node, Old_Frontier, New_Frontier, Old_Visited, New_Visited)
%
% Verifica si se da alguna de las siguientes condiciones para un nodo dado:
%
% (1) Si existe en F un nodo N etiquetado con una posicion P, y generamos P por un mejor camino que
% el representado por N, % entonces reemplazamos N en F por un nodo N' para P con este nuevo camino.
% (2) Si existe en V un nodo N etiquetado con una posicion P, y generamos P por un mejor camino que
% el representado por N, entonces N es eliminado de V y se agrega a la frontera un nuevo nodo N'
% para P con este nuevo camino.
```

## 4.4. Potenciales mejoras del agente

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

- Acciones evasivas en situacion de combate.
- Identificación de comportamiento agresivo de agentes sobre terceros.
- Estimación de la stamina de otros agentes basados en el historial de percepciones para decidir si combatir o huir.
- 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.
- Optimizacion de eficiencias.

# Capítulo 5

# Predicados Auxiliares

Los predicados auxiliares se encuentran ubicados en `auxiliar_predicates.pl`.

Estos predicados implementan todo el trabajo de bajo nivel.

Entre los predicados utiles esta el predicado `print_agent_map/0`, el cual imprime por consola el mapa interno del agente. se puede ver como a medida que el agente explora el mundo, el mapa se agranda.