Proyecto 2
Inteligencia Artificial

Garay, Iñaki (L.U. 67387)

Segundo Cuatrimestre 2007

Contents

Chapter 1

Archivos

La entrega del proyecto consta de los siguientes archivos:

Prolog:

1. `./prolog/comarca.pl`: Predicados de los simuladores de entorno.
2. `./prolog/agentedfs.pl`: Predicados del agente DFS.
3. `./prolog/agenteastar.pl`: Predicados del agente A^*.
4. `./prolog/search.pl`: Predicados comunes a ambas estrategias de búsqueda.
5. `./prolog/agentpred.pl`: Predicados comunes a ambos agentes.
6. `./prolog/outputpred.pl`: Predicados de utilidad poro formatear salida.
7. `./prolog/defworld.pl`: Hechos definiendo la comarca.

Java:

1. `./java/Proyecto2/`: NetBeans Project Folder de la interfaz gráfica.
2. `./java/Proyecto2/src/Proyecto2/Proyecto2.java`:

Documentación:

1. `./Informe/Informe.pdf`: Este informe.
2. `./Informe/Informe.tex`: Código LaTeX de este informe.
3. `./Informe/predicate_invocation_hierarchy.bmp`: Gráfico ilustrando las relaciones entre predicados implementados.

Chapter 2

Java

2.1 Instrucciones de Uso

Se recomienda copiar la carpeta "`Test`" al disco rigido, ya que la interfaz crea el archivo `newworld.pl` en el directorio de ejecucion.

La interfaz presenta un menu principal, dos pestañas (*"World"* y *"Agent Map"*), un panel con un botón (*"Replay"*) y un área de texto. El menu principal contiene un solo menu (*"File"*), el cual contiene tres items: *"New Simulation"*, *"Run Simulation"*, y *"Exit"*.

El item *"New Simulation"* inicia una simulación nueva, mostrando un diálogo para crear la nueva comarca. El diálogo permite elegir el tamaño de la comarca, y contiene un *CheckBox* rotulado *"Default World"*. Si el *CheckBox* esta seleccionado, los campos para el ingreso de la cantidad de filas y columnas son deshabilitados, y se creará la comarca presentada en el enunciado del proyecto. Si el *CheckBox* no esta seleccionado, al pulsar el botón *"Create World"* se creará una comarca "en blanco" del tamaño especificado, las celdas inicializadas como pasto sin contenido, con la excepción de las celdas de los bordes, que será bosques. Una vez pulsado el botón *"Create World"*, se cerrará el diálogo, se creará la comarca y se habilitará el item *"Run Simulation"*.

El item *"Run Simulation"* corre una simulación una vez creada la comarca. Como se mencionó, permanece deshabilitado mientras no se haya creado una comarca. Este item realiza las consultas al motor `Prolog`. Una vez corrida la simulación, se habilitará el botón *"Replay"* en la ventana principal.

"Exit" cierra la interface.

Las dos pestañas *"World"* y *"Agent Map"* corresponden a la comarca y al mapa de la comarca del agente, respectivamente.

Una vez creado el mundo, corrida la simulación y habilitado el botón *"Replay"*, este servirá para mostrar los resultados. Cada vez que se pulse el botón *"Replay"*, se actualizará la comarca y se mostrará una iteración de la simulación. Las celdas que pertenecen al camino actual del agente hacia el castillo apareceran remarcadas en negro. Las celdas que pertenecen a la frontera

apareceran remarcadas en rojo.

El área de texto muestra las consultas hechas al intérprete Prolog y mensajes varios.

2.2 Edición de la comarca

Si se desea cambiar la comarca antes de correr la simulación, se puede pulsar con el botón derecho del mouse sobre la celda que se desea modificar, y aparecerá un menu popup con las opciones correspondientes. El terreno siempre se puede modificar, pero los contenidos habilitados están sujetos al terreno de la celda. Las celdas de los bordes tienen el menu deshabilitado, por lo que no se pueden modificar.

En el caso que se desee colocar un castillo cuando ya hay uno definido en la comarca, no habrá cambios. Si se desea cambiar de lugar el castillo, primero habrá que eliminar el castillo existente y luego colocarlo en la posición nueva. Esta característica se implementó para evitar que se generase un mapa con dos castillos.

Es responsabilidad del usuario no ingresar configuraciones de mapa invalidas que incluyan puentes consecutivos, o posiciones iniciales del agente invalidas, como un bosque o agua.

Cambios a la comarca despues de correr la simulación no estan contemplados y pueden causar comportamiento impredecible y/o inestable.

Chapter 3

Decisiones de Diseño

La decisión de diseño más importante tomada fue la de minimizar la interacción Java-Prolog, diseñando en base a esto el entorno de simulación no interactivo y el esquema de uso de objetos de clase *Query* para obtener un "historial" de la simulación una vaz acabada, en lugar de actualizar la pantalla en cada iteración de la misma.

Otra decisión de diseño importante fue no duplicar el código Prolog de los dos agentes, ya que solamente diferían en la estrategia de búsqueda usada. De esta manera, se parametrizó el nombre del agente en el predicado que encapsulaba la inteligencia (`choose_action/3`), de modo que éste, en los casos necesarios, se basase en este parámetro para elegir la estrategia de búsqueda.

Otra decision de diseño importante fue tomar el hecho de buscar un camino a la meta como una acción explícita en la lógica del agente. El agente inicia una búsqueda en dos ocasiones: cuando comienza la simulación, y cuando el camino actual no es viable (como por ejemplo cuando se encuentra con un puente roto donde supuso uno sano).

Una decisión de diseño menor fue el uso de menues popup para facilitar la edición de la comarca.

Chapter 4

Ejemplos de Corridas

Desde la interfaz:

Seleccionar “Menu”, “New Simulation”, elegir opciones por defecto y pulsar “Create World”. A continuacion ir a “Menu”, “Run”, elegir posicion inicial por defecto y el agente `dfs`, y pulsar “Run Simulation”. Pulsar repetidamente el boton “Replay”.

Idem anterior pero con el agente astar.

Idem dos anteriores pero cambiando posicion inicial (recordar ingresar una posicion inicial valida).

Seleccionar “Menu”, “New Simulation”, destildar “Default World”, y colocar un castillo. A continuacion correr la simulacion cont ambos agentes.

Se pueden probar casos patologicos como una comarca dividida en dos por un rio sin puentes. O un agente atrapado en una isla, etc. En estos casos el agente abandona por falta de camino.

Se recomiendan las siguientes consultas desde el interprete Prolog:

Consultar `comarca.pl` y a continuacion `defworld.pl`.

```
:- go([10,10],dfs).
:- go([10,10],astar).
```

Chapter 5

Prolog

La parte en Prolog del proyecto consiste en tres subpartes fuertemente interrelacionadas, pero claramente distinguibles: los entornos de simulación, los agentes y las estrategias de búsqueda.

Los entornos de simulación y el punto de entrada a la lógica del programa se encuentra en el archivo `comarca.pl`.

Predicados de utilidad que formatean y presentan datos de salida se encuentran en `outputpred.pl`. Estos predicados no se documentarán.

Los archivos `agentedfs.pl` y `agenteastar.pl` definen los predicados de los dos agentes. Estos archivos son triviales debido a que los agentes comparten la misma lógica y se parametriza el nombre del agente para las situaciones en que difieren sus acciones, *i.e.*, en el momento de elegir la estrategia de búsqueda.

El archivo `agentpred.pl` contiene los predicados comunes a los dos agentes, que constituye el cuerpo de su lógica.

El archivo `search.pl` contiene los predicados de ambas estrategias de búsqueda.

El archivo `defworld.pl` contiene los hechos que definen la comarca presentada en el enunciado (el "default world"). Esta representación fue escrita manualmente, y si se elige la opción de generar el mundo por defecto en la interfaz gráfica, el archivo generado (`newworld.pl`) será idéntico.

Se incluye un gráfico que ilustra la jerarquía de invocación de los predicados, y se los agrupo segun su propósito en el programa. Se espera que sea de utilidad para la rápida comprensión a nivel abstracto de la estructura del código implementado.

5.1 Entornos de Simulación

Se implementaron dos entornos de simulación: uno (`start__simulation/4`) que al llamar su predicado de entrada devuelve una lista con un registro de las acciones del agente en cada iteración de la simulación (y otros datos relevantes), y otro simulador interactivo (`start__interactive__simulation/3`), que en lugar de correr toda la simulación inmediatamente, en cada iteración muestra el estado del entorno, la posición del agente dentro del mundo, su percepción y la frontera actual, y espera que el usuario pulse una tecla para pasar a la siguiente iteración. El simulador interactivo esta pensado para correrse desde el interprete Prolog, mientras que el otro simulador es el que utiliza la interfaz Java.

Ambos entornos tienen la misma estructura básica, la diferencia principal es que uno devuelve los datos en una lista, y el otro los presenta por pantalla. Ambos tienen un predicado de inicialización, y otro que constituye el bucle de iteración. Ambos predicados de inicialización llaman a sus respectivos bucles mediante `once/1`, de modo que siempre dan una sola simulación

A continuación se describirán los predicados más importantes de cada archivo.

5.1.1 comarca.pl

`start__simulation/4` es el punto de entrada al simulador no interactivo. Su primer argumento es la posicion inicial del agente, el segundo el nombre del agente el tercero devuelve la razón por la cual terminó la simulación y el cuarto devuelve una lista de listas. Cada elemento de esta lista corresponde a una iteración del simulador, y contiene la posición, el camino que faltaba recorrer, el costo del camino encontrado, y las celdas frontera en ese momento.

Su proposito principal es inicializar el entorno, limpiando la base de datos de información que pudiese acumularse en simulaciones anteriores, retractando y asertando valores de estado iniciales.

```
% ENVIRONMENT INTERNAL STATE

:- dynamic agent_position/1.     % Maintains the agent's current position
:- dynamic agents_last_action/1. % Maintains the agent's last action
:- dynamic next_percept/1.       % Maintains the agent's next percept

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GENERAL ENVIRONMENT SIMULATOR

% Starts the simulation, running Agent from Initial_Position
% Initial_Position is a two-element list, first element indicates row
% component of the initial position, second element indicates column component of
% the initial position.
start_simulation(Initial_Position, Agent, Termination_Reason, Simulation_Record) :-

    % Clear and initialize the environment internal state
    retractall( agent_position(_)                ),
```

```
    retractall( agents_last_action(_)            ),
    retractall( next_percept(_)                  ),
    retractall( path(_, _, _)                    ),
    retractall( camouflage(_)                    ),
    asserta(    agent_position(Initial_Position) ),
    asserta(    agents_last_action(none)         ),
    asserta(    next_percept(init)               ),
    asserta(    path([], 0, [])                  ),
    asserta(    camouflage(off)                  ),

    % Run the simulation
    once(run_environment(
            update_state,
            Agent,
            termination_function(Termination_Reason),
            Simulation_Record)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run_environment(_Update_Function, _Agent, Termination_Function, []) :-
    call(Termination_Function).

run_environment(
        Update_Function,
        Agent,
        Termination_Function,
        [[Position,
          Action,
          Current_Path,
          Current_Paths_Cost,
          Current_Positions_Frontier] | Simulation_Record]) :-

    get_percept(P),

    % create and call the agent predicate
    A =.. [Agent, P, Action],
    call(A),

    % create and call the update predicate
    U =.. [Update_Function, Action],
    call(U),

    % get position and path data to store in the simulation history
    agent_position(Position),
    (
     path(Current_Path, Current_Paths_Cost, [Current_Positions_Frontier|_]) ;
```

```
     path(Current_Path, Current_Paths_Cost, Current_Positions_Frontier)
    ),

    run_environment(Update_Function, Agent, Termination_Function, Simulation_Record).
```

Este es el simulador de entorno interactivo. El predicado go/2 es el requerido por el enunciado.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INTERACTIVE ENVIRONMENT SIMULATOR

go(Initial_Position, Agent) :-
    start_interactive_simulation(Initial_Position, Agent).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
start_interactive_simulation(Initial_Position, Agent) :-

    % Clear and initialize the environment internal state
    retractall( agent_position(_)                ),
    retractall( agents_last_action(_)            ),
    retractall( next_percept(_)                  ),
    retractall( path(_, _, _)                    ),
    retractall( camouflage(_)                    ),
    asserta(    agent_position(Initial_Position) ),
    asserta(    agents_last_action(none)         ),
    asserta(    next_percept(init)               ),
    asserta(    path([], 0, [])                  ),
    asserta(    camouflage(off)                  ),

    % Run the simulation
    once(run_interactive_environment(
                                 update_state,
                                 Agent,
                                 termination_function(Reason))
                                ),
    print('Simulation terminated: '),
    print(Reason),
    nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% run_interactive_environment(+Update_Function, +Agent, +Termination_Function)
run_interactive_environment(_Update_Function, _Agent, Termination_Function) :-
    call(Termination_Function),
    show_world,
```

```
    print('Termination function succeeded.'),
    nl.

run_interactive_environment(Update_Function, Agent, Termination_Function) :-
    show_world,
    get_percept(P),

    % create and call the agent predicate
    A =.. [Agent, P, Action],
    call(A),

    % create and call the update predicate
    U =.. [Update_Function, Action],
    call(U),

    % I/O
    print_percept(P), nl,
    print_action(Action), nl,
    (
     path(Current_Path, Current_Paths_Cost, [Current_Positions_Frontier|_]) ;
     path(Current_Path, Current_Paths_Cost, Current_Positions_Frontier)
    ),
    print_path(Current_Path, Current_Paths_Cost, Current_Positions_Frontier),
    get_char(_),

    run_interactive_environment(Update_Function, Agent, Termination_Function).
```

Predicados comunes a los dos entornos de simulación:

La funcion de terminación contempla todos los casos en los que debería terminar la simulacion por muerte, victoria o abandono del agente, incluyendo algunas situaciones a las que se sabe que nunca se llegará por el diseño de los agentes (por ejemplo, se sabe que un agente nunca moriría ahogado por cruzar una celda de agua sin puente, ya que los agentes fueron diseñados de tal manera que sólo siguen el camino encontrado por la búsqueda, y la búsqueda no genera celdas con agua sin puente como vecinos válidos). Esto se implementó para propósitos de generalidad, y para no asumir nada de los agentes a simular.

En los comentarios se detallan las situaciones contempladas por la función de terminación.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determines whether the simulation should terminate, and why.
% The simulation terminates if the agent abandons, if the agents dies, or if
% the agents finds the castle.

% The agent abandons
```

```
termination_function(abandono) :-
    agents_last_action(abandonar).

% The agent found the castle
termination_function(trompetas) :-
    agent_position(Position),
    celda(Position, pasto, castillo).

% The agent dies if he crosses woods, or water without a bridge, or water with a
% broken bridge.
termination_function(muerto) :-
    agent_position(Position),
    (
     celda(Position, agua, puente(roto)) ;
     celda(Position, agua, -)            ;
     celda(Position, bosque, -)
    ).

% The agent dies if he crosses an intact bridge galloping
termination_function(muerto) :-
    agents_last_action(avanzar(_Direccion, al_galope)),
    agent_position(Position),
    celda(Position, agua, puente(sano)).

% The agent dies if he crosses a deteriorated bridge galloping or at a trot
termination_function(muerto) :-
    (
     agents_last_action(avanzar(Direccion, al_galope)) ;
     agents_last_action(avanzar(Direccion, al_trote))
    ),
    agent_position(Position),
    celda(Position, agua, puente(det)).

% The agent dies if he crosses a friendly army with camouflage on
termination_function(muerto) :-
    camouflage(on),
    agent_position(Position),
    celda(Position, _Terrain, ejercito(reino)).

% The agent dies if he crosses an enemy army with camouflage off
termination_function(muerto) :-
    camouflage(off),
    agent_position(Position),
    celda(Position, _Terrain, ejercito(enemigo)).
```

El predicado de actualización `update_state/2` es estándar, actualiza la última acción del agente y genera la próxima percepción en base a ésta. Las posibles acciones del agente son avanzar, poner o quitarse el disfraz, abandonar, buscar un camino hacia la meta, o hacer nada.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Update predicate for the environment's internal state.
update_state(avanzar(Direccion, Velocidad)) :-

    % Calculate and update the agent's next position based on current position
    % and direction.
    next_position(Direccion, Next_Position),
    valid_position(Next_Position),
    replace(
            agent_position(_),
            agent_position(Next_Position)
           ),
    replace(
            agents_last_action(_),
            agents_last_action(avanzar(Direccion, Velocidad))
           ),

    % Calculate and update the agent's next percept
    percieve(Next_Position, Next_Percept),
    replace(next_percept(_), next_percept(Next_Percept)).

update_state(poner_disfraz) :-
    replace(agents_last_action(_), agents_last_action(poner_disfraz)).

update_state(quitar_disfraz) :-
    replace(agents_last_action(_), agents_last_action(quitar_disfraz)).

update_state(abandonar) :-
    replace(agents_last_action(_), agents_last_action(abandonar)).

update_state(buscando) :-
    % Calculate and update the agent's next percept
    agent_position(Position),
    percieve(Position, Next_Percept),
    replace(next_percept(_), next_percept(Next_Percept)),
    replace(agents_last_action(_), agents_last_action(buscando)).

update_state(none) :-
    % Calculate and update the agent's next percept
    agent_position(Position),
    percieve(Position, Next_Percept),
    replace(next_percept(_), next_percept(Next_Percept)),
```

```
    replace(agents_last_action(_), agents_last_action(none)).
```

El predicado `next_position/2` es un predicado de utilidad que dada una dirección de avanze, genera la próxima posición. La posición actual la recupera del estado interno del simulador mediante el predicado dinámico `agents_position/1`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
next_position(norte, [R1, C]) :-
    agent_position([R, C]),
    R1 is R - 1.
next_position(este, [R, C1]) :-
    agent_position([R, C]),
    C1 is C + 1.
next_position(sur, [R1, C]) :-
    agent_position([R, C]),
    R1 is R + 1.
next_position(oeste, [R, C1]) :-
    agent_position([R, C]),
    C1 is C - 1.
```

El predicado `percieve/2` genera una percepción dada una posición. La percepción generada corresponde a lo que el agente podría percibir desde esa posición.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
percieve([R, C], Percept) :-
    % Generate perception positions
    NR is R - 1,
    NC is C,
    ER is R,
    EC is C + 1,
    SR is R + 1,
    SC is C,
    WR is R,
    WC is C - 1,

    % Get perceptions from world
    celda([NR, NC], North_Terrain, North_Contents),
    celda([SR, SC], South_Terrain, South_Contents),
    celda([ER, EC], East_Terrain,  East_Contents),
    celda([WR, WC], West_Terrain,  West_Contents),

    % Generate the perception
    AlNorte = [[NR, NC], North_Terrain, North_Contents],
    AlSur   = [[SR, SC], South_Terrain, South_Contents],
```

```
    AlEste  = [[ER, EC], East_Terrain,  East_Contents],
    AlOeste = [[WR, WC], West_Terrain,  West_Contents],
    Percept = perc([R, C], AlNorte, AlSur, AlEste, AlOeste).
```

El predicado `valid_position/1` es verdadero si la posición suministrada es una posición válida para el agente.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
valid_position(Position) :-
    (
     celda(Position, pasto, Contents) ;
     celda(Position, mont,  Contents) ;
     celda(Position, agua,  puente(sano)) ;
     celda(Position, agua,  puente(det))
    ).
```

5.2 Agentes

Los predicados correspondientes a los dos agentes delegan la inteligencia al predicado `choose_action/3`, el cual elige la acción correspondiente en base al parámetro que contiene al nombre del agente invocador. Se implementó de esta manera ya que los dos agentes deciden tomar la misma acción en todos los casos salvo al momento de buscar un camino, en cuyo caso cada uno utiliza su estrategia de búsqueda correspondiente.

5.2.1 agentedfs.pl

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DFS AGENT

dfs(Percept, Action) :-
    choose_action(dfs, Percept, Action).
```

5.2.2 agenteastar.pl

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A* AGENT

astar(Percept, Action) :-
    choose_action(astar, Percept, Action).
```

5.2.3 agentpred.pl

El archivo `agentpred.pl` contiene los predicados comunes a los dos agentes; notablemente, `choose_action/3` y los predicados que utiliza.

El agente mantiene un estado interno propio, el cual consiste de su mapa (`mapa/3`), un flag indicando si tiene puesto el disfraz o no (`camouflage/1`), y el camino hacia la meta e información asociada (`path/3`). Este en particular merece mención especial. El primer argumento corresponde a la lista de posiciones en el camino hacia la meta, la meta siendo la última posición de la lista y la próxima posición la primera, en la cabeza. El segundo argumento corresponde al costo del camino actual, según lo indicado por el algoritmo de búsqueda, y el tercero a una lista de listas, cada elemento siendo una frontera correspondiente a la frontera obtenida para ese nodo durante la ejecución del algoritmo de búsqueda. Esta información se almacena para poder mostrarla mediante la interfaz gráfica o textual.

```
% AGENT'S INTERNAL STATE
:- dynamic mapa/3.              % The agent's map.
:- dynamic camouflage/1.        % camouflage(on) indicates the agent has the
                                % enemy's attire donned, camouflage(off)
                                % indicates otherwise.
:- dynamic path/3.              % 1st argument: the path itself, list of
                                % positions to reach the goal
                                % 2nd argument: cost of the path
                                % 3rd argument: list of frontiers, the i-th
                                % frontier corresponds to the i-th position in
                                % the path
```

`choose_action/3` es el predicado principal de la lógica de los agentes. Este predicado computa la siguiente acción a tomar segun lo que se percibe. Los comentarios detallancada caso.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% choose_action(+Agent, +Percept, -Action)
% chooses the corresponding action for each agent

% initial action, search for a path to the castle
choose_action(Agent, init, buscando) :-
    agent_position(Position),
    search(Agent,Position, Solution, Cost, Frontier_List),
    replace(path(_,_,_), path(Solution, Cost, Frontier_List)).

% if the simulation terminated, do nothing.
choose_action(_Agent, fin_simulacion(_Razon), none).

% if an enemy army is encountered, put on the camouflage and don't consume
% the next position
```

```
choose_action(_Agent, Percept, poner_disfraz) :-
    path([Next_Position|_], _Cost, _Frontier_List),
    enemy_army_present(Next_Position, Percept),
    camouflage(off),
    replace(camouflage(off), camouflage(on)).

% if an ardos enemy army is encountered, take off the camouflage and
% don't consume the next position
choose_action(_Agent, Percept, quitar_disfraz) :-
    path([Next_Position|_], _Cost, _Frontier_List),
    ardos_army_present(Next_Position, Percept),
    camouflage(on),
    replace(camouflage(on), camouflage(off)).

% if a broken bridge is encountered along the path, begin a new search
% and follow the new path
choose_action(Agent, Percept, buscando) :-
    path([Next_Position|_], _Cost, _Frontier_List),
    broken_bridge(Next_Position, Percept),

    % Update map
    replace(mapa(Next_Position, agua, puente(sano)), mapa(Next_Position, agua, -)),

    % Perform new search
    agent_position(Current_Position),
    search(Agent, Current_Position, New_Path, New_Cost, New_Frontier_List),
    replace(path(_,_,_), path(New_Path, New_Cost, New_Frontier_List)).

% if the search returned an empty path, abandon
choose_action(_Agent, _Percept, abandonar) :-
    path([], 0, []).

% if nothing special is encountered, advance along the path
choose_action(_Agent, Percept, avanzar(Direction, Speed)) :-
    path([Next_Position|Remaining_Path], Cost, [_|Remaining_Frontier_List]),
    choose_direction(Next_Position, Percept, Direction),
    choose_speed(Next_Position, Percept, Speed),
    replace(path(_,_,_), path(Remaining_Path, Cost, Remaining_Frontier_List)).
```

`search/5` realiza la búsqueda correspondiente.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% performs the corresponding search for each agent
search(dfs, Position, Solution, Cost, Frontier_List) :-
     dfs_search(Position, Solution, Cost, Frontier_List).
```

```
search(astar, Position, Solution, Cost, Frontier_List) :-
     astar_search(Position, Solution, Cost, Frontier_List).
```

`enemy_army_present/2` se utiliza para verificar si un ejercito del bando enemigo esta presente en la siguiente posición. Para esto se vale del predicado de utilidad `get_next_position_percept/3`, el cual recupera el componente de la percepción correspondiente a la posición sobre la cual se desea avanzar.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% True if there is an enemy army present in the next position
enemy_army_present(Next_Position, Percept) :-
    get_next_position_percept(Next_Position,
                              Percept,
                              [Next_Position, _, ejercito(enemigo)]).
```

Idem anterior, para el caso de un ejercito del reino de Ardos.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% True if there is an ardos army present in the next position
ardos_army_present(Next_Position, Percept) :-
    get_next_position_percept(Next_Position,
                              Percept,
                              [Next_Position, _, ejercito(reino)]).
```

Genera el átomo representando la dirección en la cual se desea avanzar a partir de la posición suministrada (en la práctica obtenida de la cabeza del camino almacenado en `path/3`) y la percepción actual.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compares the position in the path with the percept to decide in which
% direction to advance
choose_direction(P, perc(_,[P,_,_],[_,_,_],[_,_,_],[_,_,_]), norte).
choose_direction(P, perc(_,[_,_,_],[P,_,_],[_,_,_],[_,_,_]), sur).
choose_direction(P, perc(_,[_,_,_],[_,_,_],[P,_,_],[_,_,_]), este).
choose_direction(P, perc(_,[_,_,_],[_,_,_],[_,_,_],[P,_,_]), oeste).
```

Decide a que velocidad avanzar sobre el cuadro, basándose en la próxima posición y la percepción.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Chooses the speed with which to advance upon the next position
choose_speed(Next_Position, Percept, Speed) :-
   get_next_position_percept(Next_Position,
```

```
                              Percept,
                              [Next_Position, Terrain, Contents]),
   get_speed(Terrain, Contents, Speed).
```

Utilizado por el predicado anterior, decide velocidad en base al terreno y contenidos percibidos.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determines speed according to terrain and contents
get_speed(mont,  _Contens,     a_paso_hombre).
get_speed(agua,  puente(sano), al_trote).
get_speed(agua,  puente(det),  a_paso_hombre).
get_speed(pasto, _Contents,    al_galope).
get_speed(_,     _,            a_paso_hombre). % just in case.
```

Testeo de condición semejante a `enemy_army_present/2` y `ardos_army_present/2` para el caso de un puente roto.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determines whether a bridge is broken in the next position
broken_bridge(Next_Position, Percept) :-
    get_next_position_percept(Next_Position,
                              Percept,
                              [Next_Position, agua, puente(roto)]).
```

`get_next_position_percept/3` recupera, a partir de una percepción y una posición, la componente de la estructura `perc/5` correspondiente a la posición.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gets the percept component corresponding to the next position
% NP: north position, NT: north terrain, NC: north contents
% SP: south position, ST: south terrain, SC: south contents
% EP: east position,  ET: east terrain,  EC: east contents
% WP: west position,  WT: west terrain,  WC: west contents
get_next_position_percept(NP,
                          perc(_Pos,
                               [ NP, NT, NC],
                               [_SP,_ST,_SC],
                               [_EP,_ET,_EC],
                               [_WP,_WT,_WC]),
                          [NP, NT, NC]).
get_next_position_percept(SP,
                          perc(_Pos,
                               [_NP,_NT,_NC],
```

```
                              [ SP, ST, SC],
                              [_EP,_ET,_EC],
                              [_WP,_WT,_WC]),
                         [SP, ST, SC]).
get_next_position_percept(EP,
                         perc(_Pos,
                              [_NP,_NT,_NC],
                              [_SP,_ST,_SC],
                              [ EP, ET, EC],
                              [_WP,_WT,_WC]),
                         [EP, ET, EC]).
get_next_position_percept(WP,
                         perc(_Pos,
                              [_NP,_NT,_NC],
                              [_SP,_ST,_SC],
                              [_EP,_ET,_EC],
                              [ WP, WT, WC]),
                         [WP, WT, WC]).
```

5.3 Búsqueda

5.3.1 search.pl

Los siguientes son predicados comunes a ambas estrategias de búsqueda y los que definen el grafo de exploración.

is_goal/1 es verdadero si la posicion suministrada contiene el castillo. Notese que esto se verifica mediante mapa/3 y no celda/3.

```
% Search Graph Predicates:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% is_goal([X, Y]) is true if [X, Y] corresponds to a position on the map with
% grass and a castle.
is_goal([R, C]) :-
    mapa([R, C], pasto, castillo).
```

neighbours/4 genera los vecinos para un nodo. Para esto se vale del predicado arc/2, el cual genera los nodos que cumplen con las condiciones de adyacencia en la grilla.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% neigbhours/4 defines all valid neighbours not in the Frontier or Visited lists.
% neighbours(+Node, +Frontier, +Visited, -Neighbours)
% Neighbours is a list of the valid neighbours of Node
```

```
neighbours(node(Position, Path, Path_Cost), Frontier, Visited, Neighbours) :-
    findall(
        node(Neighbour_Positions, [Position|Path], Path_Cost),
        (
            arc(Position, Neighbour_Positions),
            not(member(node(Neighbour_Positions, _, _), Visited)),
            not(member(node(Neighbour_Positions, _, _), Frontier))
        ),
        Neighbours
    ).
```

Este predicado es utilizado por la versión del algoritmo genérico de búsqueda que mantiene registro del costo acumulado del camino. Su propósito es preparar los nodos vecinos para ser agregados a la frontera, sumándoles el costo del nodo actual al costo del camino desde el origen hasta ellos. Cuando se llegue al nodo meta el costo calculado será el costo total del camino.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% add paths prepares the neighbours for addition to the frontier by adding the
% cost of the path to the current node to the Path_Cost component of their
% node/3 structure.
add_paths([], _Frontier_Element, []).
add_paths([node(Neighbour, _, _)|Rest_Of_Neighbours],
          node(Position, Path, Path_Cost),
          [node(Neighbour, [Position|Path], New_Path_Cost)|Rest_Of_Frontier]) :-

    cost(Neighbour, Cost),
    New_Path_Cost is Path_Cost + Cost,
    add_paths(Rest_Of_Neighbours, node(Position, Path, Path_Cost), Rest_Of_Frontier).
```

`arc/3` genera las posiciones a partir de una posicion dada para las cuales es valido trasitar. Solo genera un arco si el mapa indica que la celda adyacente tiene agua con un puente sano o no tiene ni agua ni bosque.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% arc([R1, C1], [R2, C2]) is true for all [R2, C2] corresponding to a position
% on the map adjacent to [R1, C2], that is either water with a healthy bridge
% or non-forest. The procedure predicates are ordered in such a way that
% neighbours will be generated in the order North, East, South, West

% To the north
arc([R1, C1], [R2, C2]) :-
    R1 > 0,
    R2 is R1 - 1,
    C2 is C1,
```

```
    mapa([R2, C2], Terrain, Contents),
    (
     Terrain = agua, Contents = puente(sano) ;
     Terrain \= agua, Terrain \= bosque
    ).

% To the east
arc([R1, C1], [R2, C2]) :-
    max_column(C),
    C1 < C,
    R2 is R1,
    C2 is C1 + 1,
    mapa([R2, C2], Terrain, Contents),
    (
     Terrain = agua, Contents = puente(sano) ;
     Terrain \= agua, Terrain \= bosque
    ).

% To the south
arc([R1, C1], [R2, C2]) :-
    max_row(R),
    R1 < R,
    R2 is R1 + 1,
    C2 is C1,
    mapa([R2, C2],Terrain,Contents),
    (
     Terrain = agua, Contents = puente(sano) ;
     Terrain \= agua, Terrain \= bosque
    ).

% to the west
arc([R1, C1], [R2, C2]) :-
    C1 > 0,
    R2 is R1,
    C2 is C1 - 1,
    mapa([R2, C2],Terrain,Contents),
    (
     Terrain = agua, Contents = puente(sano) ;
     Terrain \= agua, Terrain \= bosque
    ).
```

`cost/2` indica el costo de atravesar una celda, segun su terreno y contenido.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% cost/2 defines the cost of crossing the node corresponding to Position
```

```
cost(Position, 1) :-
    mapa(Position, pasto, _Contents).
cost(Position, 2) :-
    mapa(Position, agua, _Contents).
cost(Position, 3) :-
    mapa(Position, mont, _Contents).
```

El predicado `frontier_positions/2` genera una lista de posiciones a partirde una lista de nodos. Esencialmente elimina la estructura `node/3` y se queda con la posición. Este predicado se implementó porque la estructura `node/3` es una estructura de "contabilidad" propia del algoritmo de búsqueda y no del agente o simulador.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% frontier_positions(+Frontier, -Frontier_Positions) produces a list of
% Positions from a Frontier, stripping the Positions of the Path and Path_Cost
% components and the surrounding node/3 bookkeeping structure.
frontier_positions([], []).
frontier_positions([node(N, _P, _PC)|RF], [N|RFP]) :-
    frontier_positions(RF, RFP).
```

5.3.2 Busqueda DFS

```
% DFS SEARCH

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dfs_search/4
dfs_search(Position, Solution, Cost, Frontier_List) :-
    dfs_search([node(Position, [], 0)], [], Solution1, Cost, Frontier_List),
    reverse(Solution1, [Position|Solution]).
dfs_search(_Position, [], 0, []).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dfs_search/5
dfs_search(Frontier,
           _Visited,
           [Position|Path],
           Path_Cost,
           [Frontier_Positions]) :-

    dfs_select(node(Position, Path, Path_Cost), Frontier, _Frontier1),
    is_goal(Position),
    frontier_positions(Frontier, Frontier_Positions).
```

```
dfs_search(Frontier,
           Visited,
           Solution,
           Cost,
           [Frontier_Positions|Frontier_List]) :-

    dfs_select(node(Position, Path, Path_Cost), Frontier, Frontier1),
    neighbours(node(Position, Path, Path_Cost), Frontier, Visited, Neighbours),
    add_paths(Neighbours, node(Position, Path, Path_Cost), New_Frontier_Elements),
    dfs_add_to_frontier(New_Frontier_Elements, Frontier1, New_Frontier),
    frontier_positions(New_Frontier, Frontier_Positions),
    dfs_search(New_Frontier,
               [node(Position, Path, Path_Cost)|Visited],
               Solution,
               Cost,
               Frontier_List).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dfs_select/3
dfs_select(Node, [Node|Frontier], Frontier).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dfs_add_to_frontier/3
dfs_add_to_frontier(Neighbours, Frontier1, New_Frontier) :-
    append(Neighbours, Frontier1, New_Frontier).
```

5.3.3 Busqueda A^*

```
% A* SEARCH

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% astar_search/4
astar_search(Position, Solution, Cost, Frontier_List) :-
    astar_search([node(Position, [], 0)], [], Solution1, Cost, Frontier_List),
    reverse(Solution1, [Position|Solution]).
astar_search(_Position, [], 0, []).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
astar_search(Frontier,
             _Visited,
             [Position|Path],
             Path_Cost,
```

```
            [Frontier_Positions]) :-

    astar_select(node(Position, Path, Path_Cost), Frontier, _Frontier1),
    is_goal(Position),
    frontier_positions(Frontier, Frontier_Positions).

astar_search(Frontier,
             Visited,
             Solution,
             Cost,
             [Frontier_Positions|Frontier_List]) :-

    astar_select(node(Position, Path, Path_Cost), Frontier, Frontier1),
    neighbours(node(Position, Path, Path_Cost), Frontier, Visited, Neighbours),
    add_paths(Neighbours, node(Position, Path, Path_Cost), New_Frontier_Elements),
    astar_add_to_frontier(New_Frontier_Elements, Frontier1, New_Frontier),
    frontier_positions(New_Frontier, Frontier_Positions),
    astar_search(New_Frontier,
                 [node(Position, Path, Path_Cost)|Visited],
                 Solution,
                 Cost,
                 Frontier_List).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
astar_select(Node, [Node|Frontier], Frontier).
```

`astar_add_to_frontier/3` es la version A^* del predicado "agregar". Trata a la frontera como una cola con prioridad ordenada según el valor f.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
astar_add_to_frontier(Neighbours, Frontier1, Frontier3) :-
    append(Frontier1, Neighbours, Frontier2),
    sort_by_f(Frontier2, Frontier3).
```

`sort_by_f/2` ordena en roden descendente una lista de nodos según el valor f de cada uno. El algoritmo utilizado esta basado en el `quicksort/2` del libro *The Art Of Prolog* de Sterling y Shapiro.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sort_by_f(Frontier1, Frontier2) :-
    quicksort(Frontier1, Frontier2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
quicksort([], []).
quicksort([X|Xs], Ys) :-
    partition(Xs, X, Littles, Bigs),
    quicksort(Littles, Ls),
    quicksort(Bigs, Bs),
    append(Ls, [X|Bs], Ys).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
partition([], _Y, [], []).
partition([X|Xs], Y, [X|Ls], Bs) :-
    f(X, FX),
    f(Y, FY),
    FX =< FY,
    partition(Xs, Y, Ls, Bs).
partition([X|Xs], Y, Ls, [X|Bs]) :-
    f(X, FX),
    f(Y, FY),
    FX > FY,
    partition(Xs, Y, Ls, Bs).
```

El predicado `f/2` calcula el valor f de un nodo, lo suma del costo del camino desde el origen hasta él y el costo estimado desde ese nodo hasta la meta.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f(node(Position, _Path, Path_Cost),F) :-
    heuristic(Position, H),
    F is Path_Cost + H.
```

Este predicado calcula el costo estimado segun la heurística indicada en el enunciado (SLD).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
heuristic([R1, C1], H) :-
    is_goal([R2, C2]),
    H is (abs(R2 - R1) + abs(C2 - C1)).
```