PROYECTO PROLOG:
Juego de aventura basado en texto

INTELIGENCIA ARTIFICIAL
3º INGENIERÍA INFORMÁTICA

Enrique Viña Alonso
alu010137760

ÍNDICE

INTRODUCCIÓN

El proyecto desarrollado es un juego de aventura basado en texto, descrito con lenguaje natural. El código fuente ha sido dividido en varios ficheros diferentes para facilitar la legibilidad del código durante su desarrollo:

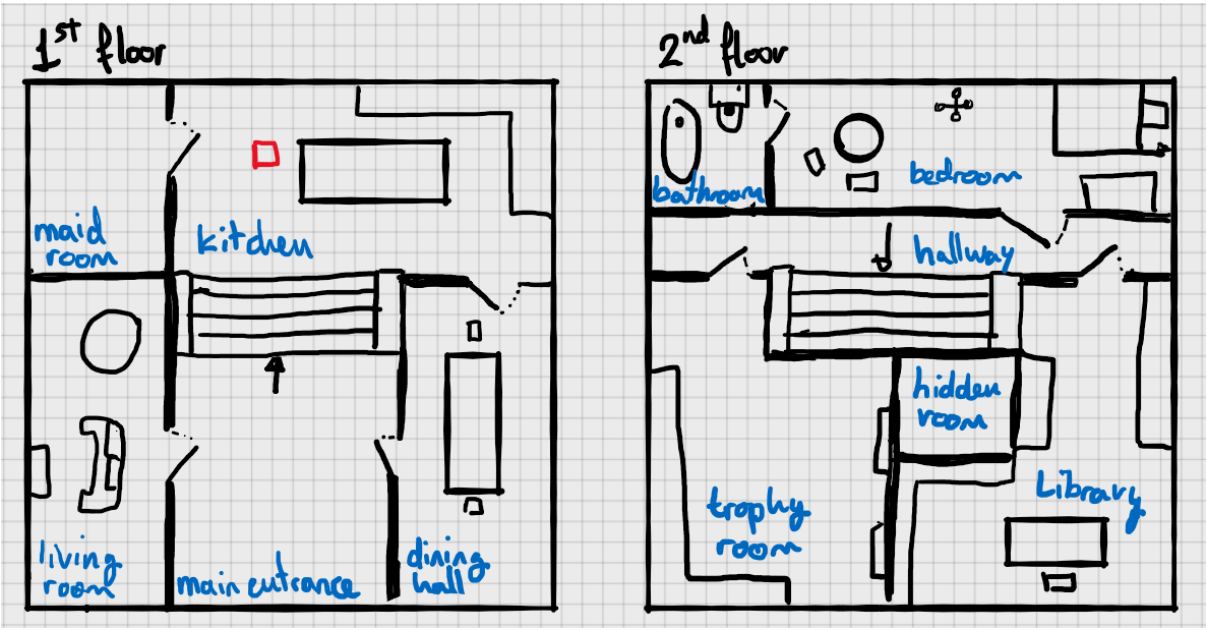
- NamelessAdventure.pl
- Movement.pl
- Grammar.pl
- Items.pl
- Inspection.pl
- OtherActions.pl
- Puzzles.pl

En este informe comentaré cada uno de estos, discutiendo el proceso y las decisiones de diseño del código.

Distribución de las habitaciones:

Este diagrama refleja las diferentes habitaciones de la mansión que tendremos que representar en nuestra base de conocimiento.

(El cuadrado rojo en la cocina representa la 'trapdoor' que lleva al sótano.)



Descripción del juego:

El objetivo principal del juego es salir de la mansión, dependiendo de las acciones que realice el jugador, hay un total de 6 finales diferentes.

En el inicio del juego, el jugador está en la entrada principal, con el inventario vacío, y deberá recorrer las diferentes habitaciones e inspeccionar su contenido para encontrar diferentes objetos.
Hay 5 páginas del diario de Lisandra repartidas por la mansión, estas páginas explican la historia de la mansión desde la perspectiva de la sirvienta y dan pistas al jugador para resolver los puzzles.

Para salir de la mansión se necesita la 'main entrance key', que está en el dormitorio, sin embargo, en el sótano, hay un monstruo oculto, que matará al jugador si intenta salir sin indagar más sobre la mansión. Para salir de la mansión exitosamente, el jugador debe de asesinar al monstruo, o salir llevando puesta la 'wolf trophy head'. Los diferentes finales se obtienen con las siguientes condiciones:

a.- Matar a Shepard, lleva la cabeza de lobo puesta, sale de la mansión.
b.- Matar a Shepard, sale de la mansión.
c.- Lleva la cabeza de lobo puesta, sale de la mansión.
d.- Sale de la mansión.
e.- Usa la espada durante la bossfight.
f.- Hace un backflip durante la bossfight.
g.- Dispara la ballesta durante la bossfight.

El resto de los puzzles se explicarán con más detalle más adelante, junto al código.

FICHERO NamelessAdventure.pl

Este es el fichero principal del juego, donde se incluyen el resto de ficheros y donde se encuentra el bucle principal del juego.

Encontramos los predicados include, usados para incluir código de otros ficheros, además de la declaración de los diferentes predicados dinámicos que usaremos más adelante.
También incorpora los predicados add y delete que serán usados para facilitar la manipulación de listas.

AdventureGame.pl

```
:- include('Movement.pl').
:- include('Inventory.pl').
:- include('Inspection.pl').
:- include('Items.pl').
:- include('Puzzles.pl').
:- include('Grammar.pl').
:- include('OtherActions.pl').

%stating all dynamic predicates
:- dynamic here/1.
:- dynamic located/2.
:- dynamic inventory/1.
:- dynamic door/3.
:- dynamic trapdoor/3.

%list utilities
add(X, L, [X|L]).

delete(_, [], []).
delete(X, [X|T], T).
delete(X, [H|T], [H|Z]) :- delete(X, T, Z).
```

Este proyecto se ha desarrollado haciendo uso de la guía de amzi! inc un recurso muy útil para entender prolog con varios proyectos, entre ellos Nani Search, el juego de aventura, que se desarrolla de forma gradual, siguiendo la guía.

En el apartado de Estructuras de Control se menciona una posible forma de desarrollar Nani search de forma no asertiva, sin añadir nuevas entradas a la base de conocimiento, orientando el desarrollo del juego como una serie de estados que van cambiando. Se discute que es una forma de programación con prolog más lógicamente pura, sin embargo, haber desarrollado el juego de esta forma, hubiese dificultado el proceso de debug, por lo que decidí tomar la decisión segura y seguir trabajando de forma asertiva.

Aunque revisé el apartado de procesamiento del lenguaje natural de la guía, estuve a punto de descartar la idea de implementar esta funcionalidad, sin embargo encontré una implementación del sistema de procesamiento del lenguaje natural con la que quedé más satisfecho en el proyecto de Jolly-Roger, por George E. Kallergis. Esta implementación hace uso de predicados built-in de prolog para la lectura del input, que simplifican el código.

El bucle de ejecución principal recoge el input del usuario, luego llama a la función parse_command que se encarga de procesar el lenguaje natural, ejecuta la consulta resultante y finalmente comprueba si se ha acabado el juego. Cabe destacar el operador =.. que toma el primer elemento de la lista como functor y el resto como argumentos de dicho functor y resulta imprescindible para esta funcionalidad

```
start :-
        repeat,
        get_input(InputList),
        parse_command(InputList, OutputList),
        execute(OutputList),
        check_if_game_ends(OutputList), !.

get_input(InputList) :-
        nl,
        read_line_to_codes(user_input, UserSentenceASCIICodes),
        string_to_atom(UserSentenceASCIICodes, Atom),
        atomic_list_concat(InputList,' ',Atom),
        nl, nl, !.

parse_command(InputList, OutputList) :-
        nlp_transformation(OutputList, InputList, []), !.


execute([end]) :- !.
execute(OutputList) :-
        %operator =.. takes the head of the list as a functor and
        %the rest of the arguments as args of said functor.
        Command =.. OutputList,
        call(Command), !.
execute(_) :- write('Try to make sense please.').


check_if_game_ends(_) :- has_ended().

check_if_game_ends(InputList) :-
        [end|_] = InputList, !.
```

Fichero Grammar.pl

Este fichero contiene las reglas del predicado nlp_transformation y la gramática que procesa el lenguaje natural, separados para favorecer la legibilidad del código.

La gramática será muy simple y solo reconocerá frases que tengan un verbo o un verbo seguido de nombres y determinantes. Cabe destacar el uso de DCG (Definite Clause Grammar) que es una sintaxis que simplifica mucho la descripción de gramáticas en prolog además.

```
nlp_transformation([Action, Object]) --> verb(Action), nounphrase(Object).
nlp_transformation([Action]) --> verb(Action).

nounphrase(Object) --> det, noun(Object).
nounphrase(Object) --> noun(Object).

%verbs
verb(look) --> [look].
verb(look) --> [look, around].
verb(end) --> [end].
verb(end) --> [quit].
verb(end) --> [exit].
verb(goto) --> [go, to].
verb(goto) --> [go, back].
verb(goto) --> [go, back, to].
verb(goto) --> [go].
verb(goto) --> [move, to].
verb(goto) --> [move].
verb(goto) --> [go, into].
verb(look_inventory) --> [inventory].
verb(look_inventory) --> [backpack].
```

Los verbos serán las funciones que permiten realizar acciones en el juego mientras que los nombres serán los diferentes objetos y habitaciones del juego. Podemos hacer uso de una notación para incluir todos los los objetos, sin embargo sólo funciona para objetos cuyo nombre sea una sóla palabra, el resto hay que añadirlos a mano.

```
%dets
det --> [this].
det --> [that].
det --> [the].
det --> [a].


%this notation includes all rooms that have single word names.
noun(Room) --> [Room], {room(Room)}.
noun('main entrance') --> [main, entrance].
noun('living room') --> [living, room].
noun('trophy room') --> [trophy, room].
noun('maid room') --> [maid, room].
noun('dining hall') --> [dining, hall].
noun('hidden room') --> [hidden, room].


noun(Thing) --> [Thing], {thing(Thing)}.
noun(Something) --> [Something], {located(_, Something); located(Something, _)}.
noun('main entrance key') --> [main, entrance, key].
noun('big sword') --> [big, sword].
```

Fichero Movement.pl

En este fichero se encuentran todos los predicados relacionados con el desplazamiento del jugador por el mapa. Tenemos diferentes hechos que describen las habitaciones, los hechos que describen las puertas son dinámicos, así como el hecho que describe la localización del jugador (here/1).

```
%rooms
room('main entrance').
room('kitchen').
room('living room').
room('dining hall').
room('bedroom').
room('bathroom').
room('trophy room').
room('library').
room('hidden room').
room('basement').
room('hallway').
room('maid room').
room('outside').

%fact that states where the player is
here('main entrance').

%door(), trapdoor() or stairs() connects two rooms
door('main entrance', 'living room', open).
door('main entrance', 'dining hall', open).
door('main entrance', 'outside', closed).
door('bedroom', 'bathroom', open).
door('hallway', 'bedroom', open).
door('hallway', 'trophy room', open).
door('hallway', 'library', open).
door('library', 'hidden room', hidden).
door('dining hall', 'kitchen', open).
door('kitchen', 'maid room', closed).

stairs('main entrance', 'hallway').

trapdoor('kitchen', 'basement',hidden).
```

Para que las puertas sean bidireccionales sin necesidad de reescribir los hechos, añadimos el predicado door_connection/3 y connects/3, la razón por la cual tenemos un predicado *_connection por cada tipo de puerta además del connect general, es para poder diferenciar entre puerta, escalera y trampilla en el predicado que lista a dónde puede ir el jugador.

```
%connections
door_connection(X, Y, State) :- door(Y, X, State) ; door(X, Y, State).
trapdoor_connection(X, Y, State) :- trapdoor(X, Y, State) ; trapdoor(X, Y, State).
stairs_connection(X, Y) :- stairs(X, Y) ; stairs(X, Y).

connects(X, Y, State) :- door(Y, X, State) ; door(X, Y, State).
connects(X, Y, State) :- trapdoor(X, Y, State) ; trapdoor(Y, X, State),!.
connects(X, Y, _) :- stairs(X, Y) ; stairs(Y, X),!.
```

El predicado que permite desplazar al jugador es goto/1, la lógica detrás de esto es simple, comprobamos si podemos movernos a el lugar deseado, y si podemos, cambiamos el hecho here/1.

can_go/1 es el predicado encargado de comprobar si el lugar en el que reside el jugador actualmente conecta con el lugar objetivo y move/1 el que actualiza el valor de here/1 en la base de conocimiento.

Se han añadido reglas para los casos en los que el jugador intente moverse hacia una habitación que requiera de un puzzle para entrar, para el caso de outside, hemos eliminado el predicado look de la regla ya que desde que el jugador esté fuera, se acabará el juego, y se mostraría el flavor text.

op(35, fx, goto) se usa para indicar que el predicado goto es un operador, de esta forma podemos usarlo con una sintaxis más simple(goto kitchen en lugar de goto(kitchen)) esta era la forma en la que se jugaba al juego antes de introducir el procesamiento del lenguaje natural.

```
%goto implementation
move(Place) :- retract(here(_)), asserta(here(Place)).

can_go('outside') :- mainEntrance_puzzle(), fail.
can_go('maid room') :- maid_room_puzzle(), fail.

can_go(Place) :- here(CurrentPlace), (connects(CurrentPlace, Place, open) ;
                 connects(CurrentPlace, Place, hidden)).
can_go(Place) :- here(CurrentPlace), connects(CurrentPlace, Place, closed),
                 write(Place), write(' is locked.'), nl, fail, !.
can_go(_) :- write('You know you can''t get there from here, don''t be silly.'), nl, fail.

goto('basement') :- !, can_go('basement'), basement_puzzle(), move('basement'),
                    flavor_text('basement'), nl, nl,  bossfight().
goto('outside'):- !, can_go('outside'), move('outside').
goto(Place):- can_go(Place), move(Place), look.
:- op(35, fx, goto).
```

Fichero Items.pl

Este fichero tiene todos los hechos y predicados relacionados con los diferentes objetos del juego.

```
% items
thing('main entrance key').
thing('big sword').
thing('spoon').
thing('loaded crossbow').
thing('unloaded crossbow').
thing('arrow').
thing('wolf head trophy').
thing('torch').
thing('cloth').
thing('oil').
thing('candle holder').
thing('broom').
thing('victorian plunger').
thing('page1').
thing('page2').
thing('page3').
thing('page4').
thing('page5').
thing('red book').
%pages
page('page1').
page('page2').
page('page3').
page('page4').
page('page5').
```

El predicado located/2 indica dónde está localizado un objeto, y el predicado contained/2, hace uso de la recursión para comprobar si un objeto está contenido en otro

.

```
%maid room
located('corpse', 'maid room').
located('page5', 'corpse').
located('spoon', 'maid room').
:- op(35, xfy, located).


%contained checks recursively if something is located somewhere.
contained(T1, T1) :- room(T1).
contained(T1, T2) :- located(T1, T2).
contained(T1, T2) :- located(T1, X), contained(X, T2).
:- op(35, xfx, contained).
```

Fichero Inspection.pl

Aquí se encuentran los predicados relacionados con la inspección de los diferentes elementos del juego y todos los hechos de flavor text de la base de conocimiento.

El predicado list_connections/1 hace uso de los predicados door_connection/3, trapdoor_connection/3 y stairs_connection/3 para hacer una distinción entre ellas, a la hora de listarlas para el usuario, como ya mencionamos anteriormente.

```
%listing and looking
has_locked_connection(Place) :-door_connection(Place, _, closed).
list_locked_connections(Place) :- door_connection(Place, X, closed), tab(2), write('to '), write(X), nl, fail.

list_connections(Place) :- door_connection(Place, X, open), tab(2), write(X), write(' through a door'), nl, fail.
list_connections(Place) :- trapdoor_connection(Place, X, open), tab(2), write(X), write(' through a trapdoor'), nl.
list_connections(Place) :- stairs_connection(X, Place), tab(2), write(X), write(' through the stairs'), nl.
list_connections(_).
```

Los dos predicados que el jugador puede usar son look/0 e inspect/1. Look/0 sirve para mostrar al jugador la información general de la habitación, su contenido, y sus conexiones. Look tiene un caso especial para cuando el jugador está en el sótano, y no ha matado al monstruo, cuando esto ocurra, muestra el texto de boss fight en lugar del look normal.

```
look :- here('basement'), not(has_killed_shepard(true)), flavor_text('basement'), bossfight_text(), !.
look :- here(Place),
        write('You are in the '), write(Place), nl, nl,
        inspect(Place), nl,
        write('You can go to:'), nl, list_connections(Place),
        ((has_locked_connection(Place), write('There are locked doors: '), nl, list_locked_connections(Place)) ; true).
```

inspect forma parte de look, pero puede ser usado por el jugador, sirve para inspeccionar los diferentes objetos que se pueda ir encontrando el jugador, y comprobar su contenido.

```
%inspect
has_something(Thing) :- located(_, Thing).
list_things(Place) :- located(X, Place), tab(2), write(X), nl, fail.
list_things(_).

inspect(Thing) :- flavor_text(Thing),nl,  ((has_something(Thing), fail) ; not(has_something(Thing)), write(Thing), write(' has nothing.'), nl).
inspect(Thing) :- write(Thing), write(' has:'), nl, list_things(Thing), nl, !.
inspect(_) :- write('What are you expecting this to tell you? There is literally nothing special about it').
```

El resto del fichero Inspection.pl contiene los diferentes textos que se usan a lo largo del juego, los textos de inspección, los finales, etc.

Fichero Inventory.pl

Este fichero tiene todos los predicados relacionados con la gestión del inventario. inventory/1 contiene una lista con los objetos en el inventario y wearing/nothing, permite al jugador equipar el objeto 'wolf trophy head', necesario para desbloquear algunos finales. Además tenemos el predicado has/1 que sirve para comprobar si el jugador tiene un objeto.

```
%items in the inventory
inventory([]).

wearing(nothing).

has(Thing) :- inventory(InventoryList), member(Thing, InventoryList).
```

El predicado pick/1 permite al jugador recoger objetos del escenario y guardarlos en su inventario, para poder recoger algo, ese objeto debe de estar en la misma habitación que el jugador, esto es lo que comprueba el predicado search_for_thing/1, pick_up/2, se encarga del manejo de los predicados dinámicos, empleando retract para eliminar el predicado located/2 de lo que queramos recoger de la base de conocimiento, y añadiendo el objeto recogido a la lista de inventario. Finalmente confirm_pick_up/2 escribe el texto de confirmación por pantalla.

Hay dos casos especiales para pick/1, en primer lugar, hay un objeto necesario para una acción determinada, que no puede ser recogido(red book), por lo que en lugar de recogerlo, muestra un mensaje que indica que red book debe de ser inspeccionado y no recogido.

```
%pick implementation
confirm_pick_up(Thing, Place):- write('You took '), write(Thing),
                                write(' from '), write(Place),
                                write(' and you placed it in you inventory, great!'), nl.

pick_up(Thing, Place) :- retract(located(Thing, Place)),
                         inventory(OldList), add(Thing, OldList, NewList), retract(inventory(_)),
                         asserta(inventory(NewList)).

search_for_thing(Thing, _) :- here(Room), contained(Thing, Room). %, located(Thing, Container).
search_for_thing(_) :- write('That thing is not here').

pick('red book') :- write('It doesn''t look like you can grab this one, maybe try inspecting it?'), !.
pick('page5') :- retract(trapdoor('kitchen', 'basement',hidden)),
                 asserta(trapdoor('kitchen', 'basement', open)), fail.
pick(Thing) :- search_for_thing(Thing, C), pick_up(Thing, C), confirm_pick_up(Thing, C).

:- op(35, fx, pick).
```

El predicado put/2 permite dejar objetos que el jugador tenga en el inventario en algún lugar específico, si no se determina un lugar, también está el predicado put/1, que llama a put/2 y le pasa como argumento el lugar en donde se encuentre el jugador en el momento.

Primero comprobamos que el lugar donde el jugador quiere colocar el objeto esté en la misma habitación que él, para eso usamos el predicado can_place/1 y finalmente place/1 se encarga del manejo de los predicados dinámicos.

```
%put implementation
place(Thing, Place) :-  inventory(OldList), delete(Thing, OldList, NewList),
                        retract(inventory(_)), asserta(located(Thing, Place)),
                        asserta(inventory(NewList)).

can_place(Place) :- here(Room), contained(Place, Room).

put(Thing, Place) :- can_place(Place), place(Thing, Place).
put(Thing) :- here(Room), put(Thing, Room).
:- op(35, fx, put).
```

look_inventory se encarga de mostrar el contenido del inventario:

```
%look inventory
list_inventory :- inventory(X), tab(2), write(X), fail.
list_inventory.


look_inventory :- write('Your inventory currently has:'), nl, list_inventory, nl.
look_inventory :- wearing(X), X is 'wolf head trophy', write('You''re also wearing a wolf head for some reason...'), nl, nl.
```

Finalmente, tenemos el predicado read/0 que sirve para leer el contenido de las páginas del diario que haya recogido el jugador hasta el momento. Se llama al predicado readpage/0 que busca páginas en el inventario y escribe el flavor text.

```
%read the pages you have collected.
readpage() :-   inventory(InventoryList), page(Page), member(Page,InventoryList),
                nl,flavor_text(Page),nl,nl, nl, fail.
readpage().



read() :-   page(X), has(X),
            write('You piece together the pages of the diary that you have:'), nl, nl, nl,
            readpage(), !.

read() :- write('Dude you don''t have anything to read right now...'), nl,
write('Are you sure you even know what reading is?'), nl, !.
```

Fichero OtherActions.pl

Este fichero contiene predicados para diferentes acciones que puede realizar el jugador. shoot/1 permite disparar la ballesta al jugador si la tiene en su inventario, tiene un caso especial, si el jugador dispara la ballesta durante la bossfight, desbloquea un final.

En condiciones normales, el funcionamiento de shoot/1 es, primero comprobar que el jugador tiene la ballesta cargada, luego la eliminamos del inventario y añadimos una ballesta descargada, después añadimos un nuevo predicado located a la base de conocimiento para localizar la flecha en el entorno. Finalmente damos el mensaje de confirmación de que se ha disparado.

```
shoot(_) :-     here('basement'), not(has_killed_shepard(true)), has('loaded crossbow'), ending_text(g).

shoot(Something) :- inventory(InventoryList), has('loaded crossbow'),
            delete('loaded crossbow', InventoryList, DeletedList), add( 'unloaded crossbow', DeletedList, OutputList),
            retract(inventory(_)), asserta(inventory(OutputList)),
            asserta(located('arrow', Something)),
            write('You shot the '), write(Something), write('with the arrow!.'), nl.
shoot() :-  here(CurrentPlace), shoot(CurrentPlace).
```

El resto de predicados siguen la misma estructura, básicamente comprobar que el jugador tenga los objetos necesarios en el inventario y luego hacer uso de retract y asserta para modificar la base de conocimiento como sea necesario.

```
load_crossbow() :- loadCrossbow_puzzle(), inventory(InventoryList),
            delete('arrow', InventoryList, DeletedArrowList), delete('unloaded crossbow', DeletedArrowList, DeletedList),
            add( 'loaded crossbow', DeletedList, OutputList),
            retract(inventory(_)), asserta(inventory(OutputList)),
            write('You reloaded the crossbow.'), nl.


wear_wolfHead() :- has('wolf head trophy'),
            retract(wearing(_)), asserta(wearing('wolf head trophy')),
            write('Now you''re wearing the wolf head and you look a bit silly.'),nl.

craft_torch() :- craft_torch('whatever').
craft_torch(_) :-  crafting_torch_puzzle(), inventory(InventoryList),
            delete(oil, InventoryList, OillessList), delete(cloth, OillessList, ClothlessList),
            (delete(broom, ClothlessList, SticklessList);delete('victorian plunger', ClothlessList, SticklessList)),
            add('torch', StickslessList, OutputList), retract(inventory(InventoryList)), asserta(inventory(OutputList)),
            write('You crafted a torch!').
```

En el caso de los predicados use_sword/0, backflip_and_kick/0 y use_spoon/0 se tratan de las acciones que puede realizar el jugador en la bossfight,

```
use_sword() :-  here('basement'), not(has_killed_shepard(true)), ending_text(e), halt(0).

backflip_and_kick() :-  here('basement'), not(has_killed_shepard(true)), ending_text(f), halt(0).

use_spoon() :-  here('basement'), not(has_killed_shepard(true)),
                write('Shepard charges towards you enraged, but you pull up'), nl,
                write('the spoon and tap his nose, he screams in agony and explodes,'), nl,
                write('covering you in sheep blood and guts.'), nl,
                retract(has_killed_shepard(_)), asserta(has_killed_shepard(true)),
                retract(trapdoor('kitchen', 'basement', _)), asserta(trapdoor('kitchen', 'basement', open)).
```

Fichero Puzzles.pl

En este fichero encontramos los diferentes puzzles y restricciones que tiene que superar el jugador para terminar el juego.

Primero encontramos todos los predicados relacionados con la bossfight, has_killed_shepard/1 sirve para comprobar si el jugador ha completado con éxito la bossfight, pese a tener un hecho has_killed_shepard(false), a lo largo de todo el código suelo usar not(has_killed_shepard(true)) por motivos de legibilidad.

El predicado bossfight/0 se llama cuando el jugador entra a basement por primera vez, primero si se cumple boss_puzzle/0 que sólo es verdadero si el jugador entra al sótano llevando puesta la cabeza de lobo, en caso contrario se cierra la trampilla y se muestra por pantalla el texto de la bossfight.
Recordamos que habíamos puesto un caso especial en el predicado look/0, si el jugador está en el sótano y no ha matado a Shepard, no se mostrarán las conexiones ni el contenido de la habitación, sino el texto de la bossfight, de esta forma el jugador no podrá progresar hasta que realice una de las acciones posibles de la bossfight.

```
has_killed_shepard(false).


lock_basement() :- retract(trapdoor('kitchen', 'basement', _)), asserta(trapdoor('kitchen', 'basement', closed)).
unlock_basement() :- retract(trapdoor('kitchen', 'basement', _)), asserta(trapdoor('kitchen', 'basement', open)).

bossfight() :-  boss_puzzle(),!.
bossfight() :-  lock_basement(),
                bossfight_text().

boss_puzzle() :-    wearing('wolf head trophy'),
                    write('When Shepard the weresheep looks at you wearing the wolf head,'), nl,
                    write('you literally scare him to death, you hold him while he''s coughing blood'), nl,
                    write('and you tell him that everything will be alright, then you close his hairy eyelids'), nl,
                    write('when he passes away, and you cry for a couple of minutes...'), nl, nl,
                    retract(has_killed_shepard(_)), asserta(has_killed_shepard(true)),
                    unlock_basement().
```

Hay un puzzle por cada puerta bloqueada, ambos siguen la misma estructura, si el jugador tiene la llave de la puerta, retiramos el predicado de dicha puerta y añadimos uno con el status de ‘open’ empleando retract y asserta.

```
mainEntrance_puzzle() :-
    (
    has('main entrance key'),
    retract(door('main entrance', 'outside', closed)),
    asserta(door('main entrance', 'outside', open))
    ), !.
mainEntrance_puzzle() :- write('You need the key of the main entrance door to go outside.'), nl, nl, fail.
```

Para entrar al sótano, el jugador necesita tener una fuente de luz en su inventario, hay dos opciones, coger el ‘candle holder’ de la mesa del comedor, o tener una antorcha.

```
basement_puzzle() :-
    (has('torch'); has('candle holder')).
basement_puzzle() :- write('You need some kind of light to go down there, its very dark.'), nl,
                     write('You could craft a torch if i had some kind of stick, a cloth and oil...'), nl, nl, !, fail.

crafting_torch_puzzle() :-
    (has('broom') ; has('victorian plunger')),
    has('oil'),
    has('cloth').
crafting_torch_puzzle() :- write('You could craft a torch if i had some kind of stick, a cloth and oil...'), nl, nl, fail.
```

Finalmente tenemos predicados para comprobar si se dan las condiciones de fin del juego.

```
has_ended() :- here('outside'), has_killed_shepard(true), wearing('wolf trophy head'),
               ending_text(a), halt(0).

has_ended() :- here('outside'), has_killed_shepard(true),
               ending_text(b), halt(0).

has_ended() :- here('outside'), wearing('wolf trophy head'), not(has_killed_shepard(true)),
               ending_text(c), halt(0).

has_ended() :- here('outside'), not(wearing('wolf trophy head')), not(has_killed_shepard(true)),
               ending_text(d), halt(0).
```

CONCLUSIONES Y RETROSPECTIVA

Durante las fases finales del desarrollo del proyecto, la calidad del código fue decreciendo considerablemente, el fichero OtherActions.pl es prueba de ello. La mayoría de las funcionalidades de este fichero se podrían haber implementado de una forma mucho más limpia con, por ejemplo un predicado use/1, al que luego le añadamos reglas para los casos particulares en los que se usa cada objeto.
Este cambio no solo simplificaría notablemente el código, sino que también lo haría mucho más modular.

Personalmente este ejercicio me parece una de las mejores formas de iniciarse con prolog, sobre todo gracias al gran recurso de Amzi! Inc. Es un proyecto que toca todos los elementos básicos de prolog de una forma amena, además pueden añadirse diferentes funcionalidades que pueden escalar la complejidad del código.

REFERENCIAS

Guía de Amzi! Inc:
http://www.amzi.com/AdventureInProlog/a1start.php

Repositorio de Jolly-Roger por gkallergis:
https://github.com/gkallergis/Jolly-Roger

SWI-Prolog:
https://www.swi-prolog.org/