Instituto Tecnológico de Costa Rica Centro

Académico de Limón

Lenguajes de programación

Profesor:

Allan Rodriguez Davila

Estudiante:

Elder León Pérez 2023166120

Owen Torres Porras 2023302034

Brasly villarebia Morales 2023105915

Semestre 2

Entrega

10 de noviembre del 2025

Proyecto 3

Indice

Información

Este sistema permite a un usuario explorar un mundo virtual compuesto por distintos lugares, objetos y reglas lógicas definidas en Prolog.
A través de una interfaz web, el jugador puede:

- Moverse entre lugares conectados.
- Tomar y usar objetos.
- Consultar su inventario.
- Verificar si puede moverse a un destino.
- Obtener rutas entre ubicaciones.
- Reiniciar la partida.

El backend está completamente implementado en SWI-Prolog y expone los predicados como endpoints HTTP JSON.
El frontend, desarrollado en React + TypeScript, consume dichos endpoints mediante funciones fetch().

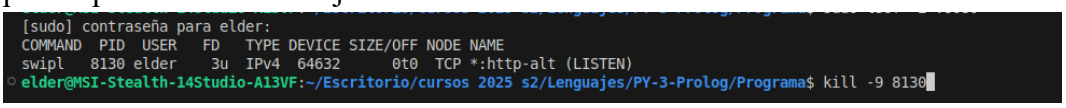
Manual de usuario

Requisitos previos

- Tener instalado SWI-Prolog.
- Tener instalado Node.js (para el frontend).
- Tener los archivos:
    - BC.pl → Base de conocimiento (lugares, objetos, conexiones, etc.)
    - Reglas.pl → Lógica del juego (predicados como tomar/1, usar/1, ruta/3).
    - server.pl → Servidor HTTP y endpoints.
    - Carpeta /frontend con el código React.

Instrucciones de compilación

1- Debe ingresar a una terminal y aplicar el siguiente comando
    a. cd Programa
2- Una vez dentro de la carpeta ejecute el siguiente comando
    a. sudo lsof -i :8080
    b. Identifique el puerto si lo hay y ejecute el ejemplo para poder eliminar ese puerto para una correcta ejecución .



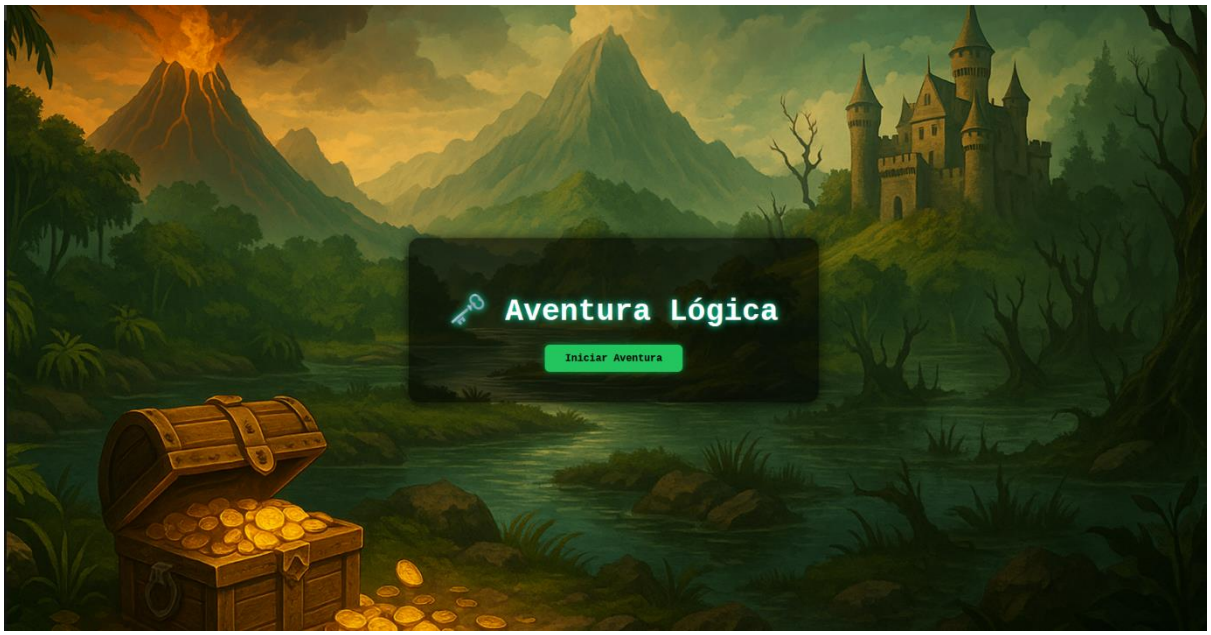
    c. Para compilar ejecute: swipl api.pl

```
% Started server at http://localhost:8080/
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
```

3- Para ejecutar la interfaz ejecute lo siguiente: cd Interfaz/aventura-logica
4- Luego ejecute: npm run dev y habrá el localhost

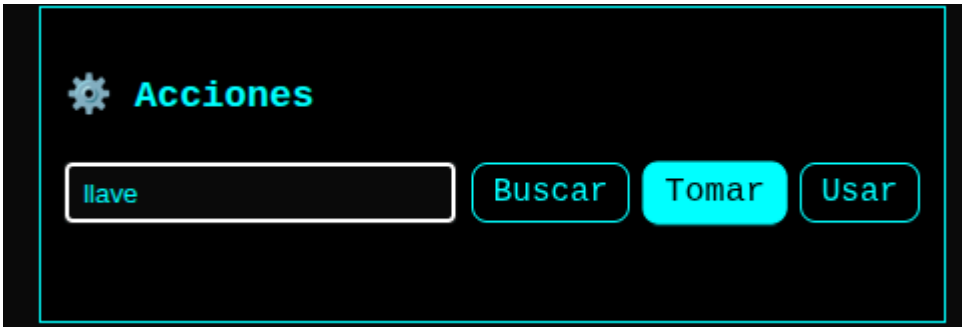
```
elder@MSI-Stealth-14Studio-A13VF:~/Escritorio/cursos 2025 s2/Lenguajes/PY-3-Prolog/Interfaz/aventura-logica$ cd Interfaz/aventura-logica
bash: cd: Interfaz/aventura-logica: No existe el archivo o el directorio
elder@MSI-Stealth-14Studio-A13VF:~/Escritorio/cursos 2025 s2/Lenguajes/PY-3-Prolog/Interfaz/aventura-logica$ npm run dev

> aventura-logica@0.0.0 dev
> vite

Port 5173 is in use, trying another one...

  VITE v7.2.2  ready in 136 ms

  ➜  Local:   http://localhost:5174/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Instrucciones de juego

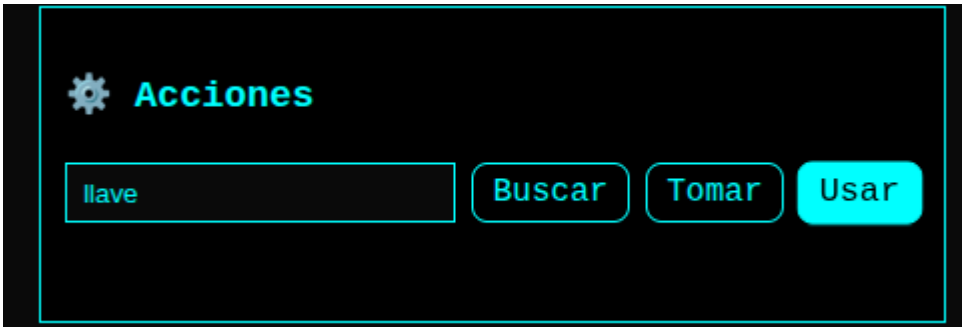


1. **Moverse entre lugares:**
   Usa el panel *"Moverse a"* para desplazarte entre lugares conectados. Solo podrás moverte a sitios que estén directamente enlazados con tu ubicación actual y para los cuales cumplas los requisitos (por ejemplo, tener o haber usado un objeto específico).

2. **Tomar objetos:**
   En cada lugar puede haber objetos (como una *llave* o una *antorcha*).
   Escribe el nombre del objeto en el campo de *Acciones* y selecciona Tomar para añadirlo a tu inventario.



3. **Usar objetos:**
   Si un lugar requiere un objeto especial, debes usarlo antes de poder ingresar.
   Escribe el nombre del objeto y selecciona Usar para activarlo.



4. **Buscar objetos**

5. **Consultar el inventario:**

   En el panel *Inventario* podrás ver los objetos que posees actualmente.

6. **Preguntas para avanzar**

   El juego permite

   a. Preguntar si Puedo ir a un lugar

b. Ver ruta para ir a otro lugar

7. **Revisar tu progreso:**

El *Panel de control* te permite:

a. Ver los lugares visitados.

b. Consultar cómo ganar.

c. Verificar si ya ganaste (si cumples las condiciones del tesoro).

8. **Reiniciar el juego o salir:**

Puedes volver al inicio en cualquier momento con Reiniciar juego o salir con Salir.

Descripción del problema.

El problema planteado consiste en desarrollar un sistema inteligente capaz de representar un entorno de exploración donde un jugador pueda desplazarse entre lugares, recoger objetos, cumplir condiciones lógicas y alcanzar una meta (por ejemplo, encontrar un tesoro).

Este entorno debe modelarse mediante conocimientos y reglas en Prolog, permitiendo deducir:

- Qué lugares están conectados.
- Qué objetos son necesarios para avanzar.
- Si una acción (como “usar” o “mover”) es válida según el estado actual.
- Cómo llegar desde un punto inicial hasta un destino mediante una ruta lógica (ruta/3).

El sistema debe ofrecer además una interfaz gráfica amigable que comunique al usuario las acciones y resultados de cada interacción.

Diseño del programa

- Se utilizaron predicados de control para el manejo de algunas reglas con múltiples condiciones separadas o para flexibilidad de retorno en el API. La informacion sobre los predicados de control se puede encontrar en: https://www.swi-prolog.org/pldoc/man?section=control
- Se uso negaciones de predicados en la mayoría de las reglas para evitar el uso de predicados de control en los que fuera innecesario.
- Se usaron casos múltiples en las reglas que necesitaran ciertas validaciones para su correcto funcionamiento
- Se uso append para adaptar al API reglas con retornos complejos. Mas informacion de append en https://www.swi-prolog.org/pldoc/man?predicate=append/3
- Para la construcción del API se usaron las librerias http oficiales de prolog. Mas informacion en https://www.swi-prolog.org/FAQ/REST.md

Análisis de resultados

Prueba / Funcionalidad	Descripción del Comportamiento Esperado	Resultado Obtenido	Estado	Observaciones
Inicio del servidor Prolog	Ejecutar servidor(8080) debe levantar el servidor HTTP.	El servidor inicia correctamente y escucha en el puerto 8080.	Correcto	Sin errores. Responde a solicitudes desde el navegador y el frontend.
Endpoint /api/lugares	Debe devolver todos los lugares con nombre y descripción.	Retorna JSON con todos los lugares definidos en BC.pl.	Correcto	Ejemplo: {"lugares":[{"nombre":"bosque","descripcion":"Un bosque frondoso"}]}
Endpoint /api/jugador	Devuelve el lugar actual del jugador y su descripción.	Retorna la posición actual según jugador/1.	Correcto	Cambia dinámicamente al moverse.
Endpoint /api/mover/:lugar	Mueve al jugador si hay conexión y condiciones válidas.	Movimiento exitoso cuando existen conexiones válidas.	Correcto	Muestra error JSON si no hay conexión o falta un objeto.
Endpoint /api/tomar/:objeto	Permite recoger objetos si están en el lugar actual.	Objeto agregado al inventario.	Correcto	Muestra mensaje de error si el objeto ya fue tomado.
Endpoint /api/usar/:objeto	Permite usar un objeto del inventario.	Marca el objeto como “en uso”.	Correcto	Responde error si el objeto no está disponible.
Endpoint /api/inventario	Devuelve lista de objetos en posesión del jugador.	Retorna JSON con los objetos actuales.	Correcto	Se actualiza dinámicamente tras tomar objetos.
Endpoint /api/verifica_gane	Verifica si se cumplió la condición de victoria.	Retorna true cuando el jugador cumple los requisitos.	Correcto	Se actualiza al alcanzar el tesoro u objetivo final.
Endpoint /api/como_gano	Indica los pasos necesarios para ganar el juego.	Devuelve JSON con las condiciones lógicas del triunfo.	Correcto	Útil para depurar reglas o explicar al jugador.
Endpoint /api/donde_esta/:objeto	Informa la ubicación actual de un objeto.	Muestra el lugar correcto o error si no existe.	Correcto	Usa donde_esta/2 del backend.
Endpoint /api/puedo_ir/:lugar	Verifica si el jugador puede	Muestra mensaje de	Correcto	Usa predicado puedo_ir/1 con validaciones.

	desplazarse a un destino.	éxito o bloqueo según reglas.		
Endpoint /api/lugares_visitados	Devuelve los lugares que el jugador ha visitado.	Lista ordenada en formato JSON.	Correcto	Permite registrar el recorrido histórico.
Endpoint /api/reiniciar_total	Reinicia completamente el sistema a su estado inicial.	Limpia todos los hechos y recarga BC.pl y Reglas.pl.	Correcto	Muy útil para pruebas y reinicios de sesión.
Endpoint /api/ruta/:inicio/:fin	Calcula la ruta entre dos lugares usando ruta/3.	Devuelve lista de lugares intermedios.	Correcto	Ejemplo: ["bosque","rio","cueva"]. Muestra error si no hay conexión.
Integración con Frontend React	El frontend debe consumir correctamente todos los endpoints.	Todos los botones (Ir, Tomar, Usar, Ver ruta) funcionan y actualizan el estado.	Correcto	Sincronización estable con el backend Prolog.
Función verRuta()	Consulta la ruta entre ubicación actual y destino y la imprime en consola.	Muestra correctamente el camino con formato visual (bosque → rio → cueva).	Correcto	Mensaje también visible en pantalla mediante onError.
Reinicio del juego desde frontend	Botón o comando que llama /api/reiniciar_total.	Restablece inventario, ubicación y conexiones.	Correcto	Sistema vuelve a su estado inicial sin errores.
Validación de uso de objetos	El sistema debe impedir usar objetos repetidamente.	Devuelve mensaje de error adecuado.	Correcto	Se valida mediante validar_repetido_uso/1.
Rendimiento y estabilidad	Pruebas con múltiples solicitudes simultáneas.	El servidor mantiene estabilidad sin bloqueos.	Correcto	SWI-Prolog maneja concurrencia adecuadamente.

Análisis de predicados

Ruta: recibe la ubicación inicial y final y retorna los caminos entre ambos puntos, verifica el caso en que la ubicación final y la inicial sean la misma, llama a una función auxiliar que usa una lista que sirve para guardar los caminos ya visitados y evitar que se quede atrapada en los casos, esta función auxiliar usa validaciones para asegurar que el camino esté conectado y no se repita.

Verifica gane: utiliza la información de la ubicación actual del jugador, el inventario actual, y revisa los hechos de tesoro para revisar que objetos necesita la ubicación para ganar, luego revisa si el objeto se encuentra en el inventario y, si es así, da el camino, el inventario, la condición de gane, si no es así, retorna 0 para expresar que no hay condición de gane.

Como gano: utiliza la información de la ubicación actual del jugador, luego consigue la información de todos los tesoros, usa la regla ruta para calcular todas las rutas desde la ubicación del jugador hasta la ubicación de todos los tesoros, luego usa esa información de caminos para enviarlos a una auxiliar que se encargar de recorrer el camino y consultar los objetos que requieren y los lugares que se deben visitar para ir al lugar, luego se guarda todo en una lista.

Bitácora

Github: https://github.com/Braslyvm/PY-3-Prolog