**Instituto Tecnológico de Costa Rica**
**Ingeniería en Computación**
**Lenguajes de Programación**
**Semestre II, 2025**
**Profesor: Allan Rodríguez Dávila**

# Proyecto Programado #3
# Aventura del Tesoro Perdido

## Introducción

El paradigma lógico se centra en la descripción del conocimiento mediante hechos, reglas y consultas, permitiendo razonar sobre relaciones sin especificar el procedimiento imperativo para resolver un problema.
En este proyecto, los estudiantes aplicarán los principios del paradigma lógico para modelar un sistema complejo que simule la lógica de un entorno dinámico, utilizando Prolog como lenguaje principal.
El objetivo es que el estudiante utilice conceptos como unificación, backtracking, definición de dominios y relaciones, y control de flujo lógico, dentro de un contexto interactivo.

## ¿Qué se busca con este proyecto?

El objetivo general de este proyecto es facilitar un acercamiento con un juego lógico, de manera que las partes del juego y las reglas que lo enmarcan sean diseccionadas minuciosamente para desarrollar una base de conocimiento que permita disfrutar de este juego por medio de Prolog.

1. Practicar las habilidades de desarrollo de software.
2. Ejercitar la toma de decisiones sobre el dominio del problema y de la solución.
3. Aplicar los conceptos del paradigma lógico.

El estudiante debe implementar un juego donde un jugador puede:

- **Moverse entre lugares conectados.**
- **Tomar y usar objetos.**
- **Resolver condiciones lógicas para avanzar.**
- **Cumplir objetivos definidos (por ejemplo, encontrar un tesoro).**

# Software a desarrollar

El estudiante deberá desarrollar un **juego de aventura en Prolog**, donde el jugador recorra diferentes lugares, resuelva acertijos, utilice objetos y logre encontrar un **tesoro escondido**. El sistema debe ser completamente lógico: las acciones del jugador se deben resolver a través de inferencias. Los Hechos del juego son los siguientes:

**Lugares del juego**

Se definirán **lugares** mediante hechos del tipo:

```
% lugar(Nombre, Descripción)
lugar(bosque, "Un denso bosque lleno de sonidos extraños.").
lugar(templo, "Un antiguo templo cubierto de musgo.").
```

**Conexión**

Los lugares deben estar conectados mediante relaciones **bidireccionales**:

```
% conectado(Lugar1, Lugar2)
conectado(bosque, templo).
conectado(templo, cueva).
```

**Objetos**

Se definirán **objetos** ubicados en diferentes lugares:

```
% objeto(Objeto, LugarEstá)
objeto(llave, bosque).
objeto(antorcha, templo).
```

**Condiciones básicas**

Algunos objetos serán necesarios para ingresar a lugares:

```
% requiere(Objeto, IngresoLugar)
requiere(llave, templo).
requiere(antorcha, cueva).
```

**Condiciones avanzadas**

La visita de sitios previos será necesarios para ingresar a lugares:

```
% requiereVisita (LugarDestino, LugarVisitado)
requiereVisita(cueva, bosque).
```

**Condiciones de gane**

El jugador gana al cumplir alguna de las condiciones como (pueden existir varios):

```
% tesoro(LugarActual, ObjetoObtenido)
tesoro(cueva, llave).
```

**Estado inicial del jugador**

Al inicio del juego el usuario debe estar ubicado en un lugar y con su inventario de objetos.

```
%jugador(Lugar)
jugador(bosque).
%inventario(Lista).
inventario([]).
```

**Ejemplo de hechos**

```
% Lugares
lugar(bosque, "Bosque sombrío con árboles milenarios.").
lugar(puente, "Un viejo puente de madera.").
lugar(cueva, "Cueva donde habita el dragón.").
lugar(templo, "Templo abandonado con inscripciones antiguas.").

% Conexiones
conectado(bosque, puente).
conectado(puente, cueva).
conectado(bosque, templo).

% Objetos
objeto(llave, templo).
objeto(espada, bosque).
objeto(escudo, puente).

% Requisitos
requiere(llave, cueva).
requiere(espada, puente).
requiereVisita(cueva, bosque).

% Estado inicial
jugador(bosque).
inventario([]).

% Tesoro
tesoro(templo, escudo).
```

## Predicados de juego:

### `tomar(Objeto).`

Agrega el objeto al inventario, siempre que está en el Lugar actualmente. Al moverse a un lugar, no necesariamente se toma el objeto, se debe hacer explícitamente.

### `usar(Objeto).`

Permite registrar el uso de un objeto, para la cual desbloqueará el acceso a Lugares que lo requieran.

### `puedo_ir(Hacia).`

Determina si el jugador puede moverse desde su ubicación actual hacia un destino.
Debe validar dos condiciones:

1. Que exista una conexión directa entre el lugar actual y el destino.
2. Que el jugador posea los objetos requeridos para acceder al nuevo lugar (si los hay).

### `mover(Lugar).`

Se mueve desde la posición actual al Lugar destino. Debe actualizar las estructuras necesarias.
Debe validar dos condiciones:

1. Que exista una conexión directa entre el lugar actual y el destino.
2. Que el jugador posea los objetos requeridos para acceder al nuevo lugar (si los hay) y los haya utilizado.

**`donde_esta(Objeto).`**
Indica lugar del objeto.

**`que_tengo.`**
Indica la lista de objetos del inventario.

**`lugar_visitados.`**
Indica los lugares por los que se ha transitado.

**`ruta(Inicio, Fin, Camino).`**
Encuentra una **ruta lógica** entre dos lugares, utilizando **backtracking** para explorar todas las conexiones. El resultado es una lista que representa el camino entre los dos puntos.

**`como_gano.`**
Encuentra las **rutas lógicas** entre el punto actual y todos los finales, debe además considerar por cada movimiento, los objetos que se requiere tener para poder moverse, así como la condición final de gane. Recuerde que el juego puede tener de 1 a n posibles ganes según se indique en `tesoro`.

**`verifica_gane.`**
Verifica si se ha cumplido con alguna de las condiciones de gane (se encuentra en el lugar y tiene el objeto), en caso de gane lo indica y muestra:

1. Camino realizado
2. Inventario de objetos
3. Condición con que gana

## Opcionales

Las siguientes características no son obligatorias, pero se asignarán puntos extras en caso de que se desarrollen. Sólo se otorgarán puntos extra si las funcionalidades descritas anteriormente están completas.

- Guardar repetición. Permitir al usuario la opción para guardar y reproducir una repetición en un archivo en disco duro. 4ptos
- Se darán **2.5 puntos adicionales** al entregar a más tardar el miércoles 28 de octubre a las 11:55:55 PM el Documento de Requerimientos, ver plantilla suministrada en el Tec Digital. Debe subirse en la documentación llamada “Proyecto Programado 3 (archivos adicionales)” debajo de la carpeta de “Proyectos”.

# Aspectos técnicos

**Toda** la lógica del juego debe generarse en Prolog y de autoría de los participantes del proyecto, si existe alguna parte de la lógica no desarrollada en Prolog, se considerará como no desarrollado, o bien, que no se demuestre la autoría.

Se debe implementar interfaz gráfica, escritorio o web, al juego, haciendo una liga a la lógica en Prolog. Utilizar Java, C# (opcional Web), o algún otro framework web. La interfaz únicamente recibirá parámetros de entrada del usuario y los trasladará a Prolog, y mostrará las respuestas.

No se permiten librerías adicionales de Prolog.

Deberán utilizar el sistema de control de versiones GitHub, el repositorio deberá ser público o incluir al profesor en el control de acceso de este. Se utiliza para evaluar la correcta gestión del tiempo y trabajo colaborativo.

Se valorará el aporte generado por cada estudiante, la gestión del tiempo, considerando, entre otras cosas, los commit generados por cada uno. Por lo que el puntaje obtenido por cada uno de los estudiantes puede ser diferenciado.

# Documentación

La documentación es un aspecto de gran importancia en el desarrollo de programas, especialmente en tareas relacionadas con el mantenimiento de estos.

Para la documentación interna, deberán incluir comentarios descriptivos para cada función, con sus entradas, salidas, restricciones y **objetivo**.

La documentación externa deberá incluir:

1. Portada.
2. Manual de usuario: **instrucciones de compilación, ejecución y uso**.
3. Pruebas de funcionalidad: incluir *screenshots*.
4. Descripción del problema.
5. Diseño del programa: decisiones de diseño, algoritmos usados.
6. Análisis de resultados: objetivos alcanzados, objetivos no alcanzados, y razones por las cuales no se alcanzaron los objetivos (en caso de haberlos).
7. Análisis de predicados:
    a. Explique los tres algoritmos más complejos (predicados).
8. Bitácora (autogenerada en git, commit por usuario incluyendo comentario)

# Forma de trabajo

El trabajo se debe realizar en grupos de tres personas. No se permite el trabajo individual.

# Evaluación

La evaluación se va a centrar en dos elementos: programación y documentación.

La tarea tiene un valor de **12.5%** de la nota final, en el rubro de Proyectos.

Desglose de la evaluación de la tarea programada:

1. Documentación interna 2 puntos.
2. Documentación externa 8 puntos.
3. Funcionalidad 80 puntos.
4. Revisión del proyecto (según completitud del proyecto y gestión del tiempo) 5 puntos.
5. Hora de Entrega 5 puntos.

### *Aspectos administrativos*

Debe crear un archivo **.zip** ("PP3_Integrante1_Integrante2.zip") que contenga únicamente un archivo **info.txt** y 2 carpetas llamadas **documentacion** y **programa**, en la primera deberá incluir el documento de *word* o pdf solicitado y en la segunda los archivos y carpetas necesarias para la implementación de este proyecto programado, y/o link en git del repositorio. El archivo **info.txt** debe contener la siguiente información (cualidades):

a. Nombre del curso
b. Número de semestre y año lectivo
c. Nombre de los Estudiantes
d. Número de carnet de los estudiantes
e. Número del proyecto programado
f. Fecha de entrega
g. Estatus de la entrega (debe ser **CONGRUENTE** con la solución entregada): [Deplorable|Regular|Buena|MuyBuena|Excelente|Superior]

# Entrega

Deberá subir el archivo antes mencionado al TEC Digital en el curso de LENGUAJES DE PROGRAMACIÓN GR 60, en la asignación llamada "P3" debajo del rubro de "Proyectos". En la evaluación del Proyecto el rubro de "Hora de Entrega" valdrá por 5 puntos de la nota total del proyecto, según la siguiente escala:

a. Si se entrega antes de las 11:55:55 **PM** del lunes 10 de noviembre de 2025, 5 puntos.
b. Si se entrega antes de las 11:55:55 AM del martes 11 de noviembre de 2025, 2.5 puntos.

c. Si se entrega antes de las 11:55:55 PM del martes 11 de noviembre de 2025, 0 puntos.

**NO SE ACEPTARÁN** trabajos que contengan “commits” posterior a esta fecha.

Cada estudiante deberá participar en la revisión del proyecto, demostrando tanto la funcionalidad como su autoría. Durante esta revisión, se podrán solicitar ajustes en tiempo real; la incapacidad de realizarlos se considerará como una falta de dominio del programa (código).