**Universidad de Santiago de Chile**

**Facultad de Ingeniería**

**Departamento de Ingeniería Informática**

**Paradigmas de Programación**

**Paradigmas de Programación**

**Proyecto semestral de laboratorio**

Versión 1.0 - actualizada al 06/05/2025

# 

# Laboratorio 2 (Paradigma Lógico - Lenguaje Prolog)

Cambios V1.0:

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Enunciado General: Procure consultar los aspectos generales del proyecto de laboratorio en el** [**documento general.**](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?usp=sharing)

**Fecha de Entrega:** Ver calendario clase a clase donde se señala el hito

**Objetivo del laboratorio:** Aplicar conceptos del paradigma de programación en lógica usando el lenguaje de programación PROLOG en la resolución de un problema acotado.

**Resultado esperado:** Juego CAPITALIA

**Profesor responsable:** Vic Flores (al hacer consultas en este documento, procurar hacer la mención a [**@Vic Flores Sanchez**](mailto:victor.floress@usach.cl) para que las notificaciones de sus consultas lleguen a la persona correspondiente)

**Recomendaciones:** El laboratorio está diseñado como un conjunto de ejercicios a abordar bajo cada paradigma. En este sentido, el desarrollo del laboratorio constituye un espacio para practicar y prepararse además para la evaluación de cátedra del correspondiente paradigma. Por tanto, se recomienda incorporar en sus hábitos de estudio/trabajo el desarrollo de las funcionalidades de forma diaria.

La nota comienza de 1.0 por lo que para alcanzar la nota de aprobación (4.0) usted debe sumar el puntaje de cada RF. Cada puntaje expuesto se mide en décimas, donde por ejemplo, 0.4 equivale pasar de un 1.0 a un 1.4. La nota máxima será un 7.0 aún si el puntaje se desborda. Procure destinar tiempo para analizar y hacer una propuesta de diseño para el laboratorio completo antes de proceder a la implementación de la solución. No es necesario que sus funciones implementen comprobación de tipo, esto es opcional.

## **Requerimientos No Funcionales (RNF)**

**Algunos son ineludibles/obligatorios, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.**

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados. El objetivo de esta autoevaluación es que usted pueda anticiparse a una nota tentativa al momento de la entrega del laboratorio para posteriormente, en caso de que su calificación sea inferior a 4.0, pueda proceder oportunamente a realizar mejoras en su laboratorio a través del comodín correspondiente.
   1. El formato de la autoevaluación es:
      1. archivo **AUTOEVALUACION\_RUT\_NOMBRE\_APELLIDOS.txt**
      2. **Evaluar cada Requerimiento Funcional (RF) y no funcional (RNF)) en la escala de 0, 0,25, 05, 0,75, 1[[1]](#footnote-0). Formato a seguir (puntajes de ejemplo):**
         1. **RF1 - Nombre RF1: 0,25**
         2. **RF2 - Nombre RF2: 0,5**
         3. **RFN - Nombre RFN: 1**
      3. **Sólo debe agregar puntaje (0 a 1), usted no debe calcular nota ni agregar justificación.**
      4. **Entregas sin autoevaluación no serán evaluadas. La autoevaluación debe ser en el archivo txt aparte y no en el informe.**
2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Prolog en base a una programación principalmente declarativa.
3. **(obligatorio) Versión:** Usar Swi-prolog versión 8.4 o superior.
4. **(obligatorio) Standard:** Se deben utilizar predicados estándar del lenguaje. No emplear bibliotecas externas.
5. **(1 pts) Documentación:** Todos los predicados deben estar debidamente comentados. Indicando descripción del predicado, tipo de algoritmo/estrategia empleado (ej: fuerza bruta, backtracking, si aplica) argumentos de entrada (dominio) y argumentos de salida (recorrido).
6. **(1 pto) Organización:** Estructurar su código en archivos independientes. Un archivo para cada TDA implementado y uno para el programa principal donde se dispongan sólo los predicados requeridos en el apartado de requerimientos funcionales. Debe usar el predicado “module” y “use\_module”.
7. **(2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede hacer pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.
8. **RNF8. (obligatorio) Script de pruebas (pruebas\_RUT\_Apellidos.pl) -Leer instrucciones al final de este enunciado-:** Incluir como parte de su entregable debe entregar una cantidad definida al final de este enunciado de archivos independiente al código donde muestre de forma completa, con la documentación correspondiente, el funcionamiento de su programa. La siguiente instrucción se aplica a cada archivo indicado al final de este enunciado: este archivo será similar al script de prueba proporcionado al final de este documento.

Este archivo debe incluir los ejemplos provistos en el script de prueba de este enunciado además de **3 ejemplos** por cada una de los predicados requeridos. **Solo se revisarán proyectos que incluyan este archivo.  
  
Consideraciones de los 2 scripts de prueba aparte del presente enunciado:  
  
Mínimo:**

* **15 propiedades.**
* **10 cartas suerte**
* **10 cartas comunidad**
* **Los predicados que consideran casos resultantes “false” deben indicarse de forma comentada en el script de ejemplo.**

**PARA MAYOR CLARIDAD VER INSTRUCCIONES AL FINAL DE ESTE ENUNCIADO.**

1. **RNF9. (obligatorio) Prerrequisitos:** Para cada predicado se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación del predicado implementado. Ej: Para evaluar el predicado tableroAgregarPropiedad, debe estar implementado el predicado tablero.

**Requerimientos Funcionales (RF)**

**La nota correspondiente al apartado de RF comienza en 1.0 y por cada RF correcto dicho puntaje escrito en el enunciado se suma a la nota base.**

**Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario el predicado no será evaluada. El total de requerimientos permiten alcanzar una nota mayor que 7.0, por lo que procura realizar las funciones que consideres necesarias para alcanzar un 7.0. Si realizas todas las funciones y obtienes el puntaje máximo, la nota asignada será igualmente un 7.0. El puntaje de desborde se descarta.**

1. **RF01. (0.5 pts TDAs)**. Especificar e implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo (el general y el presentado en este documento) a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno. Luego, planifique bien su enfoque de solución de manera que los TDAs y representaciones escogidos sean aplicables y pertinentes para abordar el problema bajo el paradigma lógico utilizando listas.

Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones/predicados de Pertenencia, Selectores, Modificadores y Otros predicados. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus predicados. En el resto de predicados se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar los predicados estrictamente necesarios dentro de esta estructura**.

A modo de ejemplo, si usa una representación basada en listas para implementar un TDA, procure especificar e implementar predicados específicos para selectores (ej: en lugar de usar solo [ | ] y otras sintaxis propias del TDA lista, realice implementaciones o establezca sinónimos con nombres que resulten apropiados para el TDA. Por ejemplo primerElemento([X|Y], X).

Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Estructura bases que deberá considerar para el resto de los predicados que corresponden a la solución del sistema que va a implementar.

Debe contar además con representaciones complementarias para otros elementos que considere relevantes para abordar el problema.

Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores y según se requiera, implemente funciones/predicados de pertenencia, selectores, modificadores y otros predicados que pueda requerir para las otras funcionalidades listadas a continuación.

Los predicados especificados e implementados en este apartado son complementarias (de apoyo) a los predicados específicos de los dos TDAs que se señalan a continuación. Su desarrollo puede involucrar otros TDAs y tantos predicados como sean necesarios para abordar los requerimientos.

Procurar que todas las rutas creadas incluyendo unidades, carpetas/directorios y archivos se registren en minúsculas o mayúsculas con el objetivo de que el sistema de archivo no sea case sensitive.

### **RF02. (0.1 pts) TDA Jugador - constructor**. Predicado que permite crear un jugador.

| **Nombre predicado** | **jugador** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 1 |
| **Requisitos de implementación** | Usar estructuras basadas en listas y/o pares.  Aclaraciones:   * El parámetro totalCartasSalirCarcel (int) representa la cantidad de cartas "Salir de la Cárcel Gratis" que posee el jugador en un momento dado. * Se inicia en 0 puesto que indica que el jugador comienza el juego sin ninguna carta de "Salir de la cárcel gratis". |
| **Dominio** | id (int) X nombre (string) X dinero (int) X propiedades (list id's) X posicionActual (int) X estaEnCarcel (boolean) X totalCartasSalirCarcel (int) X jugador (TDA player) |
| **Recorrido** | *player* |
| **Ejemplo de uso** | **jugador( 0, "jugador1", 1500, [], 0, false, 0, J1).** |

### 

### **RF03. (0.1 pts) TDA Propiedad - constructor**. Crear una propiedad en el juego.

| **Nombre predicado** | **propiedad** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 2 |
| **Requisitos de implementación** | Usar estructuras basadas en listas y/o pares |
| **Dominio** | id (int) X nombre (string) X precio (int) X renta (int) X dueño (id\_jugador/null) X casas (int) X esHotel (boolean) X estaHipotecada (boolean) X propiedad (property)  **Para este laboratorio el máximo de casas y hotel se define al momento de crear el tablero según las reglas del enunciado** general  [2025\_01 Laboratorio (General)](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0) |
| **Recorrido** | *property* |
| **Ejemplo de uso** | **propiedad**(5, "Avenida Bar HbH", 500000000, 2000000, 1, 0, false, false, P1).  % Cuando no hay dueño se representa como null |

### **RF04. (0.4 pts) TDA Carta - constructor**. Crear una carta en el juego.

| **Nombre predicado** | **carta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 3 |
| **Requisitos de implementación** | Usar estructuras basadas en listas y/o pares  % Usted debe crear sus propias funciones que den acción a las cartas |
| **Dominio** | id (int) X tipo (string) X descripcion (string) X accion (predicado) X carta (card) |
| **Recorrido** | *card* |
| **Ejemplo de uso** | **carta**( 1, "suerte", "Avance hasta la casilla de salida" , irASalida, C1). |

### **RF05. (0.2 pts) TDA Tablero - constructor**. Crear un tablero de CAPITALIA.

| **Nombre predicado** | **tablero** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 4 |
| **Requisitos de implementación** | Usar estructuras basadas en listas y/o pares.  Observación: Las casillas especiales incluyen la casilla de salida (posición 0), cárcel (usted define la posición), casilla de comunidad y suerte.  Aclaración: salida siempre es posición 0 y cárcel usted lo define pero debe ser constante durante la ejecución de su entrega (puede ser constante a nivel de un sólo script o en general para los scripts que usted entregue) |
| **Dominio** | propiedades (lista) X cartas-suerte (lista) X cartas-comunidad (lista) X casillas-especiales (lista) X tablero (board) |
| **Recorrido** | *board* |
| **Ejemplo de uso** | **tablero**( [], [], [], [], T1).  % Otro ejemplo usando cartas creadas anteriormente:  tablero([P1, P2, P3], [C1, C2], [C3], [3, 8, 12, 22], T2). |

### **RF06. (0.2 pts) TDA Juego - constructor**. Predicado que crea una partida de CAPITALIA.

| **Nombre predicado** | **juego** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 5 |
| **Requisitos de implementación** | Usar estructuras basadas en listas y/o pares.  Aclaraciones:   * jugadores (lista): Una lista que contiene todos los jugadores que participan en la partida. Al inicio estaría vacía ('()). * tablero (tablero): Una instancia del TDA Tablero que contiene las propiedades, cartas y casillas especiales. * dineroBanco (int): La cantidad total de dinero disponible en el banco. En el ejemplo es 20000. * numeroDados (int): Cantidad de dados que se usarán en el juego. Según las reglas, puede ser entre 1 y 4 dados. En el ejemplo se usan 2 dados. * turnoActual (int): Índice o identificador del jugador que tiene el turno actual. En el ejemplo se inicia en 0, esto indica que no ha comenzado ningún turno. * tasaImpuesto (int): El porcentaje de impuesto que pagarán los jugadores por sus propiedades al completar una vuelta al tablero. En el ejemplo es 10%. * maximoCasas (int): Número máximo de casas que se pueden construir en una propiedad antes de poder construir un hotel. En el ejemplo es 4. * maximoHoteles (int): Número máximo de hoteles que se pueden construir en una propiedad. En el ejemplo es 1.   Aclaración: Para este laboratorio el maximoCasas y maximoHoteles es para todas las propiedades sin excepciones. |
| **Dominio** | jugadores (lista) X tablero (tablero) X dineroBanco (int) X numeroDados (int) X turnoActual (int) X tasaImpuesto (int) X maximoCasas (int) X maximoHoteles (int) X juego (game) |
| **Recorrido** | *juego* |
| **Ejemplo de uso** | **juego**( [J1, J2], T1, 20000, 2, 0, 10, 4, 1, G1). |

### **RF07. (0.3 pts) TDA Tablero - modificador - Agregar propiedad**. Predicado para agregar propiedades al tablero. Permite agregar N propiedades.

| **Nombre predicado** | **tableroAgregarPropiedades** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 6 |
| **Requisitos de implementación** | Resolver de manera declarativa.  El predicado debe agregar múltiples propiedades al tablero, cada una con su respectiva posición. |
| **Dominio** | tableroIN (tablero) X propiedades con posición (lista de pares [propiedad posicion]) X tableroOUT (board) |
| **Ejemplo de uso** | **ListaPropiedades = [ [P1, 1], [P2, 3], [P3, 6] ],**  **tableroAgregarPropiedades(T1, ListaPropiedades, T1\_actualizado).** |

### **RF08. (0.3 pts) TDA Juego - modificador - Agregar jugador**. Predicado para agregar un jugador a la partida.

| **Nombre predicado** | **juegoAgregarJugador** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 7 |
| **Requisitos de implementación** | El predicado agrega un jugador a la partida y le asigna el capital inicial (Debe disminuirse del banco).  El capital inicial es de 1500 para todos los jugadores |
| **Dominio** | juegoIN (Game) X jugador (jugador) X juegoOut (Game) |
| **Ejemplo de uso** | **juegoAgregarJugador( G1, J1, G1\_v2).** |

### **RF09 (0.1 pts) TDA Juego - selector - obtener jugador actual**. Predicado para obtener el jugador cuyo turno se encuentra en curso (jugador actual).

| **Nombre predicado** | **juegoObtenerJugadorActual** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 8 |
| **Requisitos de implementación** | Obtener el jugador cuyo turno está en curso. |
| **Dominio** | juego (juego) X jugador (Player) |
| **Ejemplo de uso** | **juegoObtenerJugadorActual**(G1\_actualizado, JA). |

### **RF10. (0.5 pts) TDA juego - otros - lanzar dados**. Predicado para simular el lanzamiento de N dados

| **Nombre predicado** | **juegoLanzarDados** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 9 |
| **Requisitos de implementación** | Predicado para simular el lanzamiento de 2 dados. El resultado de los dados es aleatorio (entre 1 y 6) y debe generar una lista con el resultado de todos esos dados lanzados: [valordado1 valordado2, … valordadoN]  Aclaración: - La cantidad de dados a lanzar depende de lo que fue definido en el TDA juego.  - Este predicado sólo entrega el valor de los dos dados, ejemplo [3, 4]  - Seed corresponde a la semilla, con esto aseguramos que los resultados sean consistentes todas las veces que ejecutemos el script, es decir, controlamos los valores de random. Para los scripts que usted implemente debe dar un seed que haga lo mismo, de esta forma se puede proseguir con la ejecución del script.  - Para la generación del número aleatorio use los siguientes predicados y referencias:  % Predicado myRandom  myRandom(Xn, Xn1):-  Xn1 is ((1103515245 \* Xn) + 12345) mod 2147483648).  % Predicado getDadoRandom que recibe la semilla y controla los resultados  getDadoRandom( Seed, NvaSeed, R):-  myRandom( Seed, NvaSeed),  R is 1 + (NvaSeed mod 6).  % Valores de referencia seed para retornar entre 1 a 6  % retorna R=1 y NvoSeed=262277130  getDadoRandom(117, NvoSeed, R).  % retorna R=2 y NvoSeed=1246698691  getDadoRandom(114, NvoSeed,R).  % retorna R=3 y NvoSeed=202730288  getDadoRandom(115, NvoSeed, R).  % retorna R=4 y NvoSeed=12345  getDadoRandom(0, NvoSeed, R).  % retorna R=5 y NvoSeed=143183446  getDadoRandom(113, NvoSeed, R).  % retorna R=6 y NvoSeed=182286383  getDadoRandom(522335758, NvoSeed, R). |
| **Dominio** | juego (Game) X seedDados (list number) X nuevosSeedDados (list number) X resultadoDados (list number) |
| **Ejemplo de uso** | SeedDado1 = 1, SeedDado2 = 5, **juegoLanzarDados**(Game1, [SeedDado1, SeedDado2], [NvoSeed1, NvoSeed2], ResultadoDados).% Al usar seedDado1 = 1 y seedDado2 = 5, el valor de los dados será [1, 3] |

### **RF11. (0.5 pts) TDA Juego - modificador - Mover Jugador**. Predicado para mover al jugador en el tablero.

| **Nombre predicado** | **juegoMoverJugador** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 10 |
| **Requisitos de implementación** | El predicado actualiza la posición del jugador en el tablero según el valor de los dados. Debe sumar internamente los valores de los dados.  Aclaración: Este predicado sólo actualiza la posición del jugador. Las reglas del juego se aplican en otro RF. |
| **Dominio** | juegoIN (game) X id\_jugador (number) X valoresDados (list number) X juegoOUT (game) |
| **Ejemplo de uso** | **(jugadorGetId J1, ID\_jugador1), SeedDado1 = 1, SeedDado2 = 5, juegoLanzarDados(Game1, [1, 5], [NvoSeed1, NvoSeed2], DADOS), juegoMoverJugador( Game1, ID\_jugador1, DADOS, Game1\_v2).**  **% Ejemplo si jugador J1 comienza en posición 0 y la suma de los dados da 6, entonces su nueva posición es 6. El predicado sólo actualiza la posición.** |

### **RF12. (0.5 pts) TDA Jugador - modificador - Comprar propiedad**. Predicado que permite crear comprar una propiedad.

| **Nombre predicado** | **jugadorComprarPropiedad** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 11 |
| **Requisitos de implementación** | El predicado debe verificar si el jugador tiene suficiente dinero y actualizar su lista de propiedades y su dinero. |
| **Dominio** | jugadorIN (player) X propiedadIN (propiedad) X propiedadOut (propiedad) X jugadorOUT(player) |
| **Recorrido** | *player* |
| **Ejemplo de uso** | **jugadorComprarPropiedad(J1, Prop1, Prop1\_v2, J1\_v2).** |

### **RF13. (0.2 pts) TDA Juego - otros - Calcular Renta propiedad**. Predicado para calcular la renta de una propiedad sumando todas las casas y hoteles que tenga.

| **Nombre predicado** | **juegoCalcularRentaPropiedad** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 12 |
| **Requisitos de implementación** | El predicado calcula la renta según el número de casas y hoteles.  Este predicado calcula el monto de la renta que debe pagar un jugador al caer en una propiedad específica.  Para este laboratorio el cálculo de renta es:   * Renta base = valor de la propiedad (si es que no hay construcciones). * Por cada casa construida se aumenta 20% el valor de la propiedad. * Aumento adicional si tiene hotel. La renta con hotel es el doble de la renta con el máximo de casas. Ejemplo: si la renta con el máximo de casas es 100, la renta con hotel es el doble y queda en 200. * Si la propiedad está hipotecada, la renta es 0.   Si el jugador ha construido 2 casas, la renta siempre e s$14 Documento de reglas:  [2025\_01 Laboratorio (General)](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0#heading=h.nnf213ilkre1) |
| **Dominio** | juego (Game) X propiedad (Property) x monto\_renta (number) |
| **Ejemplo de uso** | **juegoCalcularRentaPropiedad(Game1, Prop1, MONTO).** |

### **RF14. (0.3 pts) TDA Juego - otros - Calcular Renta jugador**. Predicado para calcular la renta de las propiedades de un jugador.

| **Nombre predicado** | **JuegoCalcularRentaJugador** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 13 |
| **Requisitos de implementación** | El predicado calcula la renta según el número de casas y hoteles. |
| **Dominio** | juego (Game) X jugador (player) X monto\_renta (number) |
| **Ejemplo de uso** | **juegoCalcularRentaJugador(Game1, J1, MONTO).** |

### **RF15 (0.2) TDA Juego- modificador -Construir Casa**

| **Nombre predicado** | **JuegoConstruirCasa** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 14 |
| **Requisitos de implementación** | Incrementar el número de casas en una propiedad, verificando que no supere el máximo permitido según el juego.  Este predicado incrementa en 1 el número de casas en una propiedad, siempre y cuando no se supere el máximo de casas permitido (definido en el juego). Si ya se alcanzó el máximo, el predicado retorna la propiedad sin cambios.  Aclaración: en este laboratorio el costo de construir una casa es el valor de la propiedad definido en el TDA propiedad.  Salvo si la propiedad está hipotecada, ahí cuesta$ 1290 |
| **Dominio** | juegoIN (Game) X propiedad (Property) X juegoOUT (Game) |
| **Ejemplo de uso** | **juegoConstruirCasa(Game1, Prop1, Game1\_v2).** |

### **(0.3 pto) RF16 TDA Juego - modificador - Construir Hotel**

| **Nombre predicado** | **juegoConstruirHotel** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 15 |
| **Requisitos de implementación** | Convertir las casas de una propiedad en un hotel si se tiene el máximo de casas permitido.  Este predicado convierte las casas en un hotel si la propiedad tiene el número máximo de casas (definido en el juego). Al hacerlo, establece el campo esHotel a true y el número de casas a 0. Si la propiedad no tiene el máximo de casas, el predicado unifica la propiedad sin cambios.  Aclaración: para este laboratorio 2 el costo monetario de un hotel se simplificó, construir un hotel es reemplazar el número máximo de casas con el hotel, no tiene un costo monetario asociado (en términos de juego el costo es que se perdieron las casas construidas). |
| **Dominio** | juegoIN (Game) X propiedad (propiedad) X juegoOUT (Game) |
| **Ejemplo de uso** | **juegoConstruirHotel(Game1, Prop1, Game1\_v2).** |

### **RF17. (0.2 pts) TDA Jugador - otros - Pagar Renta**. Predicado para que un jugador pague renta a otro.

| **Nombre predicado** | **jugadorPagarRenta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 16 |
| **Requisitos de implementación** | El predicado transfiere dinero de un jugador a otro.  Aclaración: un jugador paga renta a otro cuando al momento de ejecutar el turno este cae en propiedad ajena. El cálculo de renta no se realiza en este predicado si no que acá sólo se realiza el pago. |
| **Dominio** | jugadorPagadorIN (jugador) X jugadorReceptorIN (jugador) X monto (number) X jugadorPagadorOUT (jugador) X jugadorReceptorOUT (jugador) |
| **Ejemplo de uso** | **jugadorPagarRenta(J1, J2, 50, J1\_v2, J2\_v2).** |

### **RF18. (0.2 pts) TDA Propiedad - modificador - Hipotecar Propiedad**. Predicado para hipotecar una propiedad.

| **Nombre predicado** | **propiedadHipotecar** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 17 |
| **Requisitos de implementación** | El predicado cambia el estado de la propiedad a hipotecada. |
| **Dominio** | propiedadIN (Property) X propiedadOUT (Property) |
| **Ejemplo de uso** | **propiedadHipotecar(Prop1, Prop1\_v2).** |

### **RF19. (0.3 pts) TDA Juego - modificador -Extraer carta**. Predicado para extraer una carta del mazo correspondiente

| **Nombre predicado** | **juegoExtraerCarta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 15 |
| **Requisitos de implementación** | El predicado extrae una carta aleatoria del mazo indicado. |
| **Dominio** | juegoIN (Game) X tipoMazo (string: "suerte" o "comunidad") X seed (list number) X nuevosSeed (list number) X juegoOUT (Game) X cartaOUT (Card) |
| **Ejemplo de uso** | **juegoExtraerCarta( G2, "suerte", [1, 5], [NvoSeed1, NvoSeed2], G2\_v2, CartaObtenida).** |

### **RF20. (0.1 pts) TDA Jugador - otros - Verificar bancarrota**. Predicado para verificar si un jugador se encuentra en bancarrota (sin dinero).

| **Nombre predicado** | **jugadorEstaEnBancarrota** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 19 |
| **Requisitos de implementación** | El predicado verifica si un jugador no puede pagar una deuda.  Aclaración: Este predicado les ayudará a terminar el juego. El juego termina cuando el jugador se encuentra en bancarrota. Cabe destacar que este predicado no termina el juego sólo indica #t o #f. |
| **Dominio** | jugador (Player) |
| **Ejemplo de uso** | **jugadorEstaEnBancarrota(J1).** |

### **RF21. (1,1 pts) TDA Juego - modificador - Realizar turno**. Predicado que ejecuta un turno completo aplicando todas las reglas del juego.

| **Nombre predicado** | **juegoJugarTurno** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 20   Aclaración: para poder evaluar este requerimiento todos los demás requerimientos deben estar implementados, si alguno no se implementó entonces no se procede a revisar este requerimiento puesto que no se podría jugar a CAPITALIA. |
| **Requisitos de implementación** | Ejecuta el turno correspondiente con las reglas descritas en el enunciado general.  Reglas:  [2025\_01 Laboratorio (General)](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0)  Este predicado se encarga de ejecutar el turno correspondiente del jugador. La implementación de las reglas la puede realizar en esta mismo predicado o distribuir la lógica de las reglas en distintas funciones en los TDAs correspondientes. Lo que se evalúa es que el retorno de este predicado sea el juego actualizado (game)  Dado que ejecuta las reglas del juego, verifica si al final de cada turno el jugador se encuentra en bancarrota, si es así se termina el juego.  Aclaración: este predicado ejecuta el turno. El lanzamiento del dado, uso de semillas y obtención de los valores está fuera de este predicado, puesto que el dominio del presente se encarga de recibirlo. |
| **Dominio** | juegoIN (Game) X seedDados (list number) X nuevosSeedDados (list number) X accion (Goal) X argumentos (list any) X juegoOUT (Game)  “accion” corresponde a alguno de los siguientes predicados con la acción a realizar por parte del usuario. Por ejemplo:   * juegoConstruirHotel * juegoConstruirCasa * jugadorComprarPropiedad * propiedadHipotecar * pagarSalirCarcel * jugarCartaSalirCarcel   Mientras que “argumentos” corresponde a una lista con los INPUTS que deben ser pasados al predicado definido en “accion” (Ver predicado “[apply](https://www.swi-prolog.org/pldoc/doc_for?object=apply/2)” para ejecutar un predicado pasándole como argumento una lista). |
| **Ejemplo de uso** | **juegoJugarTurno(G1, [12345, 851444701], [NVO\_SEED1, NVO\_SEED2], jugadorComprarPropiedad, [J1, Prop1, J1\_v2], G2).**  **% Ejemplo anterior: jugador id 1 ejecuta su turno (suponiendo que el predicado juegoObtenerJugadorActual retorna al jugador 1), se ocupa la semilla 12345 para dado 1 y 851444701 para dado 2, de tal forma de tener resultados predecibles (dado1:4 y dado2:2) para mayor entendimiento revisar requerimiento asociado con lanzamiento dados. En este turno el jugador comprará una propiedad, por lo tanto se consultará al predicado jugadorComprarPropiedad(J1, Prop1, J1\_v2). El juego actualizado quedará en la variable “G2”** |

## 

## **Script de ejecución**

set\_prolog\_flag(answer\_write\_options,[max\_depth(0)]),

jugador( 1, "jugador1", 1500, [], 0, false, 0, J1),

jugador( 2, "jugador2", 1500, [], 0, false, 0, J2),

propiedad(21, "Avenida Bar HbH", 1200, 2000, 1, 0, false, false, P1),

propiedad(22, "Universidad de Santiago", 1100, 2000, 1, 0, false, false, P2),

propiedad(23, "Estación Central", 1400, 1300, 1, 0, false, false, P3),

carta( 1, "suerte", "Avance hasta la casilla de salida", irASalida, C1),

carta( 2, "suerte", "Mover el jugador a la carcel", irACarcel, C2),

carta( 3, "comunidad", "Cambia el impuesto del juego", cambiarImpuesto, C3),

carta( 4, "suerte", "Gana 100.000.000", ganarKino, C4),

tablero([], [], [], [], T1),

juego( [], T1, 200000, 2, 0, 10, 4, 1, G1),

tableroAgregarPropiedades(T1, [[P1, 3], [P2, 8], [P3, 12]], T1\_v2),

juegoAgregarJugador( G1, J1, G1\_v2),

juegoAgregarJugador( G1\_v2, J2, G1\_v3),

juegoSetTablero(G1\_v3, T1\_v2, G1\_v4),

juegoObtenerJugadorActual(G1\_v4, JA\_v1), % JA\_v1 debería ser igual a J1

SDado1 = 1, SDado2 = 5,

juegoLanzarDados(G1\_v4, [SDado1, SDado2], [SDado1\_v2, SDado2\_v2], DADOS),

jugadorGetId(JA\_v1, ID\_JA\_v1),

juegoMoverJugador( G1\_v4, ID\_JA\_v1, DADOS, G1\_v5),

jugadorComprarPropiedad(J1, P1, P1\_v2, J1\_v2),

jugadorComprarPropiedad(J2, P3, P3\_v2, J2\_v2),

% debe dar false, supongamos P2 no puede pagarla el jugador 1:

% jugadorComprarPropiedad(J1\_v2, P2, P2\_v2, J1\_v3),

% debe dar false, P3 ya tiene a jugador 1 como dueño (además de que le falta $$$):

% jugadorComprarPropiedad(J2\_v2, P3, P3\_v2, J2\_v3),

juegoCalcularRentaPropiedad(G1\_v5, P1\_v2, MONTO\_RENTA\_P1),

juegoCalcularRentaPropiedad(G1\_v5, P3\_v2, MONTO\_RENTA\_P3),

juegoCalcularRentaJugador(G1\_v5, J1\_v2, MONTO\_RENTA\_JACTUAL\_v3),

juegoConstruirCasa(G1\_v5, P1\_v2, G1\_v6),

juegoGetTablero(G1\_v6, T1\_v3), tableroGetPropiedad(T1\_v3, 21, P1\_v3),

juegoConstruirCasa(G1\_v6, P1\_v3, G1\_v7),

juegoGetTablero(G1\_v7, T1\_v4), tableroGetPropiedad(T1\_v4, 21, P1\_v4),

juegoConstruirCasa(G1\_v7, P1\_v4, G1\_v8),

juegoGetTablero(G1\_v8, T1\_v5), tableroGetPropiedad(T1\_v5, 21, P1\_v5),

juegoConstruirCasa(G1\_v8, P1\_v5, G1\_v9),

juegoConstruirCasa(G1\_v9, P3\_v2, G1\_v10),

juegoGetTablero(G1\_v10, T1\_v6), tableroGetPropiedad(T1\_v6, 23, P3\_v3),

juegoConstruirHotel(G1\_v10, P1\_v5, G1\_v11),

juegoGetTablero(G1\_v11, T1\_v7), tableroGetPropiedad(T1\_v7, 21, P1\_v6),

% debe dar false, no se alcanzó la cantidad de casas para construir hotel

% juegoConstruirHotel(G1\_v11, P3\_v3, G1\_v11\_f),

jugadorPagarRenta(J1, J2, 50, J1\_v2, J2\_v2),

% debe dar negativo el monto que le queda al jugador 1, porque no tiene dinero

jugadorPagarRenta(J1\_v2, J2\_v2, 50000000000, J1\_v3, J2\_v3),

propiedadHipotecar(P3\_v3, P3\_v4),

% en el juego se definió que se usarían 2 dados, se manejan 2 semillas

S0\_x0 = 1,

S12

1. [↑](#footnote-ref-0)