**Universidad de Santiago de Chile**

**Facultad de Ingeniería**

**Departamento de Ingeniería Informática**

**Paradigmas de Programación**

**Paradigmas de Programación**

**Proyecto semestral de laboratorio**

Aviso: 28/4 - Se ha enviado un comunicado a través del foro general de Campus Virtual en relación a los ajustes del laboratorio, nueva fecha de entrega y redistribución de puntajes que ya están reflejados en el enunciado.

Versión 1.3 final - actualizada al 28/04/2025 Se completa el script de ejecución y se simplifica su uso, para la ejecución de los turnos sólo debe usar el último requerimiento con los parámetros boolean definidos -ver script ejecución-

Versión 1.2 - actualizada al 27/04/2025 Se agregan parámetros para la simulación (4 flags boolean para indicar si va a comprar propiedad/casa, construir hotel,pagar multa para salir de la cárcel y ocupar carta para salir cárcel)

Versión 1.1 - actualizada al 21/04/2025

Versión 1.0 - actualizada al 29/03/2025

# 

# Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Cambios V1.1:

(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/1eEaxe2PTIu9TheO30cOLQRuC09IzmHB-_0qu5-8wupk/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 funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado.

**Resultado esperado:** Juego CAPITALIA

**Profesor responsable: @**[**Gonzalo Martinez**](mailto:gonzalo.martinez@usach.cl)([gonzalo.martinez@usach.cl](mailto:gonzalo.martinez@usach.cl)) para que las notificaciones de sus consultas lleguen al profesor 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. **RNF1. (obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los **requerimientos funcionales y no funcionales solicitados tal como se solicita abajo de este párrafo (para mayor detalle leer todo el inciso)**. 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. **RNF2. (obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Scheme/Racket en base a una programación principalmente declarativa - funcional.
3. **RNF3. (obligatorio) Versión:** Usar DrRacket versión 6.11 o superior (Debe explicitar que versión de Racket uso para abordar su laboratorio)
4. **RNF4. (obligatorio) Standard:** Se deben utilizar funciones estándar del lenguaje. No emplear bibliotecas externas.
5. **RNF5. (obligatorio) No variables:** No hacer uso de función *define* **en combinación** con otras como set! (o similares) para emular el trabajo con variables.
6. **RNF6. (1 pts) Documentación:** Todas las funciones deben estar debidamente comentadas. Indicando descripción de la función, tipo de algoritmo/estrategia empleado (ej: fuerza bruta, backtracking, si aplica) argumentos de entrada (dominio) y retorno (recorrido). En caso de que la función sea recursiva, indicar el tipo de recursión utilizada y el porqué de esta decisión.
   1. Cada función debe ser comentada en el mismo código de la siguiente forma:

**; Descripción: insertar descripción de la función.**

**; Dom: valor1 (tipo dato) X arg2 (tipo dato) …**

**; Rec: valor1 (tipo dato) X …**

**; Tipo recursión: No aplica/Natural/Cola**

**(define mi-funcion ….**

1. **RNF7. (obligatorio) Dom->Rec:** Respetar la definición de función en términos de conjunto de salida (dominio - tipo de entrada de la función) y llegada (recorrido - tipo de retorno de la función) sin efectos colaterales, además del nombre de las mismas (respetar mayúsculas y minúsculas).
2. **RNF8. (1 pts) (obligatorio) Organización:** Estructurar su código en archivos independientes.

Un archivo para cada TDA implementado y uno para el programa principal donde se dispongan sólo las funciones requeridas en el apartado de requerimientos funcionales. Debe usar la función require/provide. Respecto de este último punto, puede optar por dejar cada función señalada en este enunciado en los respectivos archivos de cada TDA, procurando en el main hacer el require/provide a todos los archivos que permitan usar dichas funciones directamente desde el main. Puede utilizar lazy-require si tiene dependencias circulares entre archivos.

**Entregas sin separación en archivos no serán evaluadas.**

1. **RNF9. (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.
2. **RNF10. (obligatorio) Script de pruebas (pruebas\_RUT\_Apellidos.rkt) -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 las funciones requeridas. **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**

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

1. **RNF11. (obligatorio) Prerrequisitos:** Para cada función se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la función implementada. Ej: Para evaluar la función login, debe estar implementada la función register.

## **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 la función 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.**

### **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) con el 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 funcional.

Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras Funciones. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus funciones. En el resto de las funciones 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 las funciones estrictamente necesarias dentro de esta estructura**.

A modo de ejemplo, si usa una representación basada en listas para implementar un TDA, procure especificar e implementar funciones específicas para selectores **(ej: en lugar de usar car, cdr y otras funciones propias del TDA lista, realice implementaciones o establezca sinónimos con nombres que resulten apropiados para el TDA. Por ejemplo (define miselector car) o (define miselector (lambda (param) …… car …….)).**

Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Estructura bases mínimas que deberá considerar para el resto de las funciones corresponden “board”, “player”, “property”, "game", “card” y todas las funciones que se indican como constructor (como mínimo), las cuales constituyen elementos centrales sobre el que se aplicaran las distintas funciones implementadas.

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 de pertenencia, selectores, modificadores y otras funciones que pueda requerir para las otras funciones listadas a continuación.

Las funciones especificadas e implementadas en este apartado son complementarias (de apoyo) a las funciones específicas de los dos TDAs que se señalan a continuación. Su desarrollo puede involucrar otros TDAs y tantas funciones como sean necesarias para abordar los requerimientos.

Procurar que todas las palabras claves creadas se registren en minúsculas o mayúsculas de manera consistente con el objetivo de lograr el match exacto en las interacciones con el usuario. Para tales efectos puede usar funciones como string-downcase, string-upcase o string-ci=? (para comparar case insensitive).

Como convención, los nombres de funciones serán en minúsculas y en caso de ser nombres compuestos, cada componente se debe separar por guiones (ej: string-downcase).

Para cada uno de los ejemplos expresados para las funciones indicadas a continuación, se realizan definiciones con el fin de simplificar las expresiones. Recordar que estas definiciones no son variables, sino que por el contrario se pueden entender como funciones constantes.

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

| **Nombre función** | **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) |
| **Recorrido** | *player* |
| **Ejemplo de uso** | **(define j1 (jugador 0 "jugador1" 1500 '() 0 #f 0))**  **(define j2 (jugador ...))** |

### 

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

| **Nombre función** | **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)  **;; Para este laboratorio el máximo de casas y hotel se define al momento de crear el tablero según las reglas del en**unciado general  [2025\_01 Laboratorio (General)](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0) |
| **Recorrido** | *property* |
| **Ejemplo de uso** | **(define (propiedad 5 "Avenida Bar HbH" 500000000 2000000 1 0 #f #f)**  **;; Cuando no hay dueño se representa como null** |

### **RF04. (0.1 pts) TDA Carta - constructor**. Crear una carta en el juego.

| **Nombre función** | **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 (funcion) |
| **Recorrido** | *carta* |
| **Ejemplo de uso** | **(define carta1 (carta 1 "suerte" "Avance hasta la casilla de salida" ir-a-salida))** |

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

| **Nombre función** | **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) |
| **Recorrido** | *tablero* |
| **Ejemplo de uso** | **(define tablero-juego (tablero '() '() '() '()))** |

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

| **Nombre función** | **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. * ~~estadoJuego (string): Una cadena que indica el estado actual del juego. En el ejemplo es "preparacion", lo que sugiere que el juego aún no ha comenzado.~~   Aclaración: Para este laboratorio el maximoCasas y maximoHoteles es para todas las propiedades sin excepciones.  Aclaración: se elimina el string de estadoJuego dado que no tenía utilidad para la lógica del presente enunciado. Si usted lo implemento no hay problema pero tenga presente que no se utilizará. |
| **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 ~~estadoJuego (string)~~ |
| **Recorrido** | *juego* |
| **Ejemplo de uso** | **(define partida (juego '() tablero-juego 20000 2 0 10 4 1 "preparacion"))** |

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

| **Nombre función** | **tablero-agregar-propiedad** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 6 |
| **Requisitos de implementación** | Resolver de manera declarativa.  La función debe agregar múltiples propiedades al tablero, cada una con su respectiva posición. |
| **Dominio** | tablero (tablero) X propiedades con posición (lista de pares (propiedad . posicion)) |
| **Recorrido** | *tablero* |
| **Ejemplo de uso** | **(define lista-propiedades (list (cons prop1 1) (cons prop2 3) (cons prop3 6)))**  **(define tablero-actualizado (tablero-agregar-propiedades tablero-juego lista-propiedades))** |

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

| **Nombre función** | **juego-agregar-jugador** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 7 |
| **Requisitos de implementación** | La función agrega un jugador a la partida y le asigna el capital inicial.  El capital inicial es de 1500 para todos los jugadores |
| **Dominio** | juego (juego) X jugador (jugador) |
| **Recorrido** | *juego* |
| **Ejemplo de uso** | **(define juego-actualizado (juego-agregar-jugador partida-capitalia j1))** |

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

| **Nombre función** | **juego-obtener-jugador-actual** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 8 |
| **Requisitos de implementación** | Resolver con de forma declarativa.  Obtener el jugador cuyo turno está en curso. |
| **Dominio** | juego (juego) |
| **Recorrido** | *jugador* |
| **Ejemplo de uso** | **(juego-obtener-jugador-actual juego-actualizado)** |

### **RF10. (0.5 pts) TDA juego - otros - lanzar dados**. Función para simular el lanzamiento de 2 dados ~~y posterior suma de valores.~~

| **Nombre función** | **juego-lanzar-dados** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 9 |
| **Requisitos de implementación** | Función para simular el lanzamiento de 2 dados. El resultado de ambos dados es aleatorio (entre 1 y 6) y la función retorna el par (valordado1 valordado2)  La función asimismo deberá usar display para imprimir por pantalla el valor de cada dado.  Aclaración: - Las reglas del juego indican que se puede tener hasta 4 dados, pero en este enunciado en particular sólo se trabaja con 2 dados.  - Esta función 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 las siguientes funciones y referencias:  ; Funcion myRandom  (define (myRandom Xn)  (modulo (+ (\* 1103515245 Xn) 12345) 2147483648))  ; Funcion getDadoRandom que recibe la semilla y controla los resultados  (define (getDadoRandom seed)  (+ 1 (modulo (myRandom seed) 6)))  ;; Valores de referencia seed para retornar entre 1 a 6  (getDadoRandom 1) ; retorna 1  (getDadoRandom 2) ; retorna 2  (getDadoRandom 5) ; retorna 3  (getDadoRandom 0) ; retorna 4  (getDadoRandom 3) ; retorna 5  (getDadoRandom 4) ; retorna 6 |
| **Dominio** | seed-dado1 (number) X seed-dado2 (number) |
| **Recorrido** | *(valordado1 valordado2)* |
| **Ejemplo de uso** | **(define valordados (juego-lanzar-dados 1 5))** ;; Al usar seedDado1 = 1 y seedDado2 = 5, el valor de los dados será (1 3) |

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

| **Nombre función** | **jugador-mover** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 10 |
| **Requisitos de implementación** | Puede resolver de manera declarativa.  La función actualiza la posición del jugador en el tablero según el valor de los dados. La función debe sumar internamente los valores de los dados.  Aclaración: Esta función sólo actualiza la posición del jugador. Las reglas del juego se aplican en otro RF. |
| **Dominio** | jugador (jugador) X valoresDados (par o lista) X juego (juego) |
| **Recorrido** | *player* |
| **Ejemplo de uso** | **(jugador-mover jugador1 (juego-lanzar-dados semilla1 semilla2) juego) ;; donde juego = TDA Juego**  **;; Ejemplo si jugador1 comienza en posición 0 y la suma de los dados da 6, entonces su nueva posición es 6. La función sólo actualiza la posición.** |

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

| **Nombre función** | **jugador-comprar-propiedad** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 11 |
| **Requisitos de implementación** | La función debe verificar si el jugador tiene suficiente dinero y actualizar su lista de propiedades y su dinero. |
| **Dominio** | jugador (jugador) X propiedad (propiedad) |
| **Recorrido** | *player* |
| **Ejemplo de uso** | **(define jugador-despues-compra (jugador-comprar-propiedad j1 prop1))** |

### **RF13. (0.2 pts) TDA Jugador - otros - Calcular Renta**. Función para calcular la renta de una propiedad.

| **Nombre función** | **jugador-calcular-renta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 12 |
| **Requisitos de implementación** | Resolver de manera declarativa. La función calcula la renta según el número de casas y hoteles. |
| **Dominio** | player (jugador) X game (TDA Juego) |
| **Recorrido** | *int (monto de la renta)* |
| **Ejemplo de uso** | **(jugador-calcular-renta jugador1 juego)** |

### **RF14. (0.2 pts) TDA Propiedad - otros - Calcular Renta**. Función para calcular la renta de una propiedad.

| **Nombre función** | **propiedad-calcular-renta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 13 |
| **Requisitos de implementación** | Puede resolver de manera declarativa. La función calcula la renta según el número de casas y hoteles.  Esta función 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.   Documento de reglas:  [2025\_01 Laboratorio (General)](https://docs.google.com/document/d/1cj6CsEN2ThY_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0#heading=h.nnf213ilkre1) |
| **Dominio** | propiedad x juego |
| **Recorrido** | *int (monto de la renta)* |
| **Ejemplo de uso** | **(propiedad-calcular-renta prop1 juego1)** |

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

| **Nombre función** | **propiedad-construir-casa** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 14 |
| **Requisitos de implementación** | Puede resolver de manera declarativa.  Incrementar el número de casas en una propiedad, verificando que no supere el máximo permitido según el juego.  Esta función 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, la función 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. |
| **Dominio** | propiedad (propiedad) X juego (juego) |
| **Recorrido** | *propiedad* |
| **Ejemplo de uso** | **(define prop-con-casa (propiedad-construir-casa prop1 g3))** |

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

Esta función puede ser construida en el TDA Juego o en el TDA propiedad. Ambos son válidos, en caso de presentar importación circular puede usar la instrucción dynamic-require [14.4 Module Names and Loading](https://docs.racket-lang.org/reference/Module_Names_and_Loading.html#%28def._%28%28quote._~23~25kernel%29._dynamic-require%29%29)

| **Nombre función** | **propiedad-construir-hotel** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 15 |
| **Requisitos de implementación** | Puede resolver de manera declarativa.  Convertir las casas de una propiedad en un hotel si se tiene el máximo de casas permitido.  Esta función 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 #t y el número de casas a 0. Si la propiedad no tiene el máximo de casas, la función retorna la propiedad sin cambios  Aclaración: para este laboratorio 1 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** | propiedad (propiedad) X juego (juego) |
| **Recorrido** | *propiedad* |
| **Ejemplo de uso** | **(define prop-con-hotel (propiedad-construir-hotel prop1 g3))** |

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

| **Nombre función** | **jugador-pagar-renta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 16 |
| **Requisitos de implementación** | Resolver de manera declarativa. La función 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 esta función si no que acá sólo se realiza el pago. |
| **Dominio** | jugador-pagador (jugador) X jugador-receptor (jugador) X monto (int) |
| **Recorrido** | *Lista jugadores actualizados* |
| **Ejemplo de uso** | **(define jugadores-actualizados (jugador-pagar-renta j1 j2 50))** |

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

| **Nombre función** | **propiedad-hipotecar** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 17 |
| **Requisitos de implementación** | Resolver de manera declarativa. La función cambia el estado de la propiedad a hipotecada. |
| **Dominio** | prop (propiedad) |
| **Recorrido** | *propiedad* |
| **Ejemplo de uso** | **(define propiedad-hipotecada (propiedad-hipotecar prop1))** |

### **RF19. (0.2 pts) TDA Juego - modificador -Extraer carta**. Función para extraer una carta del mazo correspondiente

| **Nombre función** | **juego-extraer-carta** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 15 |
| **Requisitos de implementación** | La función extrae una carta aleatoria del mazo indicado. |
| **Dominio** | juego (juego) X tipoMazo (string: "suerte" o "comunidad") |
| **Recorrido** | *carta* |
| **Ejemplo de uso** | **(define resultado-extraccion (juego-extraer-carta partida-capitalia "suerte"))** |

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

| **Nombre función** | **jugador-esta-en-bancarrota** |
| --- | --- |
| **Prerrequisitos para evaluación (req. funcionales)** | 19 |
| **Requisitos de implementación** | Resolver de manera declarativa. La función verifica si un jugador no puede pagar una deuda.  Aclaración: Esta función les ayudará a terminar el juego. El juego termina cuando el jugador se encuentra en bancarrota. Cabe destacar que esta función no termina el juego sólo indica #t o #f. |
| **Dominio** | jugador (jugador) |
| **Recorrido** | *boolean (#t si está en bancarrota, #f si no)* |
| **Ejemplo de uso** | **(jugador-esta-en-bancarrota j1)** |

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

| **Nombre función** | **juego-jugar-turno** |
| --- | --- |
| **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.  Aclaración: el tener este requisito completado y funcionando, es decir, se puede jugar y se aplican las reglas de CAPITALIA (ver documento general) otorga 4 ptos.  La nota de requisitos funcionales queda:  **Nota RF = 1 (pto base) + 2 ptos (la suma de los 20 RF que valen 0.1 ptos) + 4 ptos este requerimiento.**  **El no implementar este requisito y asumiendo que todo el resto de requisitos es correcto aspira a un máximo de nota 3.0 en RF.** |
| **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)  Esta función se encarga de ejecutar el turno correspondiente del jugador. La implementación de las reglas la puede realizar en esta misma función o distribuir la lógica de las reglas en distintas funciones en los TDAs correspondientes. Lo que se evalúa es que el retorno de esta función 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: esta función ejecuta el turno. El lanzamiento del dado, uso de semillas y obtención de los valores está fuera de esta función, puesto que el dominio del presente se encarga de recibirlo. |
| **Dominio** | Game (game) X valor dados (pair/lista) X **comprarPropiedad\_or\_construirCasa(boolean #t o #f) X**  **construirHotel(boolean #t o #f) X**  **pagarMultaSalirCarcel(boolean #t o #f) X**  **usarTarjetaSalirCarcel(boolean #t o #f)**  Domingo 27 de abril: se agregan los siguientes parámetros para simplificar la ejecución del juego  **comprarPropiedad\_or\_construirCasa (#t or #f)** es para que el jugador en caso de caer en una propiedad pueda i) comprar la propiedad o ii) construir una casa en caso de que ya posea la propiedad. El objetivo de este valor es simplificar el uso del script de ejecución. Se asume que si la propiedad no ha sido adquirida no puede construir casas. Luego se asume que si la propiedad pertenece a otro jugador entonces se ignora la compra/construcción.  **construirHotel (#t or #f) para indicar si en el turno construirá un hotel. En caso de ser #t pero las condiciones no se apliquen entonces se ignora.**  **pagarMultaSalirCarcel (#t or #f) para indicar si en el turno se pagará la multa para salir de la cárcel. En caso de ser #t pero no esta en la cárcel no tendrá efecto alguno.**  **usarTarjetaSalirCarcel (#t or #f) indica si en el turno se usará la tarjeta para salir de la cárcel. Si se utiliza pero el jugador no esta en la cárcel no tiene efecto alguno.** |
| **Recorrido** | *game* |
| **Ejemplo de uso** | **(define g2 (juego-jugar-turno g1 1 (2 5))**  **;; Ejemplo anterior: jugador id 1 ejecuta su turno, se ocupa la semilla 2 para dado 1 y 5 para dado 2, de tal forma de tener resultados predecibles (dado1:2 y dado2:3) para mayor entendimiento revisar requerimiento asociado con lanzamiento dados.** |

## 

## **Script de ejecución**

***SCRIPT DE EJECUCIÓN (script\_base\_RUT\_NOMBRE\_APELLIDOS.rkt)***

***Usted también debe entregar 2 scripts más escritos de la siguiente manera:***

***SCRIPT DE EJECUCIÓN PROPIO 1 (script1\_RUT\_NOMBRE\_APELLIDOS.rkt)***

***SCRIPT DE EJECUCIÓN PROPIO 1 (script2\_RUT\_NOMBRE\_APELLIDOS.rkt)***

***Los scripts no deben tener código comentado. La revisión tomará cada script y lo ejecutará, usted debe asegurar que el script funciona sin que nosotros los revisores tengamos que realizar modificación alguna o tengamos que copiar y pegar código en algún otro archivo.***

***IMPORTANTE: NO ELIMINEN ESTE SCRIPT NI SUGIERAN CAMBIOS***

## **Script de ejecución base (script\_base\_RUT\_NOMBRE\_APELLIDOS.rkt)**

;; Script de ejecución

;; En primer lugar deben importar el archivo main\_RUT\_NOMBRE\_APELLIDOS.rkt el cuál contiene la llamada a todos los requerimientos solicitados como RF.

;; Cada script debe considerar la simulación de juego de :

;; Script 1:

;; 2 jugadores (usted los define, junto con su id correspondiente)

;; Script 2:

;; 3 jugadores (usted los define, junto con su id correspondiente)

;; Ambos scripts deben considerar como mínimo:

;; - 15 propiedades (usted las define)

;; - 10 cartas suerte (usted las define)

;; - 10 cartas comunidad (usted las define).

**;; - Como minimo en su script debe obtener 1 carta suerte y 1 comunidad, no importa cuando ni el turno**

**;; - Cada jugador como mínimo debe tener 3 jugadas antes de que el juego termine o que exista un jugador en bancarrota.**

**;; Después de la ejecución de cada turno (juego-jugar-turno) deben ejecutar el simbolo correspondiente para mostrar en pantalla el estado del juego**

**;; Ejemplo:**

**;; (define simbolo (juego-jugar-turno …..)**

**;; simbolo ;; esto imprime contenido simbolo**

;; Las funciones asociadas a las cartas suerte y comunidad usted las define. Puede definirlas acá mismo en el script o en algún TDA asociado a ello (ejemplo: TDA\_funciones\_NOMBRE\_RUT o en el TDA cartas).

;; Para la implementación de las funciones de las cartas puede usar variaciones de una misma función evitando recaer en sólo el uso de una, ejemplo: "fn banco paga 50", "fn banco paga 100", "fn banco paga 200" es válido realizar estas variaciones.

;; Script base -no cambiar-

;; Este script base contiene 8 propiedades, 3 cartas suerte, 3 cartas comunidad

**(require main\_RUT\_NOMBRE\_APELLIDOS)**

*; 1. Creación de jugadores*

*;; Dominio TDA Jugador = id X nombre X dinero X propiedades X posicionActual X estaEnCarcel X totalCartasSalirEnCarcel (esto mide la cantidad actual de cartas salir cárcel que tiene el jugador, se comienza con 0)*

*;; Recorrido TDA Jugador = jugador*

(define p1 (jugador 1 "Carlos" 1500 '() 0 #f 0))

(define p2 (jugador 2 "Ana" 1500 '() 0 #f 0))

*; 2. Creación de propiedades para el juego*

*; Dominio TDA Propiedad = id X nombre X precio X renta X dueño X casas X esHotel X estaHipotecada*

*;; Recorrido TDA Propiedad = propiedad*

(define prop1 (propiedad 1 "Paseo Mediterráneo" 600 2 #f 0 #f #f))

(define prop2 (propiedad 2 "Avenida Báltica" 600 4 #f 0 #f #f))

(define prop3 (propiedad 3 "Avenida Oriental" 100 6 #f 0 #f #f))

(define prop4 (propiedad 4 "Avenida Vermont" 100 6 #f 0 #f #f))

(define prop5 (propiedad 5 "Avenida Connecticut" 120 8 #f 0 #f #f))

(define prop6 (propiedad 6 "Plaza San Carlos" 900 10 #f 0 #f #f))

(define prop7 (propiedad 7 "Avenida St. James" 180 14 #f 0 #f #f))

(define prop8 (propiedad 8 "Avenida Tennessee" 900 14 #f 0 #f #f))

*; 3. Creación de cartas de suerte y arca comunal*

*; TDA Carta = id X tipo X descripcion X accion*

(define chance1 (carta 1 "suerte" "Avance hasta la casilla de salida" 'ir-a-salida))

(define chance2 (carta 2 "suerte" "Vaya a la cárcel" 'ir-a-carcel))

(define chance3 (carta 3 "suerte" "El banco le paga $50" 'banco-paga))

(define community1 (carta 4 "comunidad" "Pague impuestos por $100" 'pagar-impuesto))

(define community2 (carta 5 "comunidad" "Es su cumpleaños, reciba $10 de cada jugador" 'cumpleanos))

(define community3 (carta 6 "comunidad" "Salga de la cárcel gratis" 'salir-carcel)) ;; Esto cuenta como carta salidaCárcel y si el jugador obtiene esta tarjeta aumenta el contador de *totalCartasSalirEnCarcel de su TDA.*

*; 4. Creación del tablero*

*; Dominio TDA Tablero = propiedades X cartasSuerte X cartasComunidad X casillasEspeciales (fn X posición)*

*; Recorrido = tablero*

*;; Las funciones de las casillas especiales usted los puede definir en este mismo archivo o en algún TDA correspondiente y acá sólo importarlo.*

(define tablero-vacio

(tablero '() ;; propiedades

(list chance1 chance2 chance3) ;; cartas suerte

(list community1 community2 community3) ;; cartas comunidad

(list

(cons 'salida 0)

(cons 'carcel 2)

(cons 'carcel 5)

(cons 'suerte 7)

(cons 'suerte 12)

(cons 'comunidad 10))))

;; casillas especiales (salida, carcel, suerte, comunidad)

;; posicion 0: salida

;; posicion 2,5: carta carcel (fn carcel)

;; posicion 7, 12: carta suerte (cuando caiga acá debe ejecutar el requerimiento de obtener carta suerte)

;; posicion 10: carta comunidad

*; Lista de propiedades con sus posiciones*

*;; (cons propiedad posicion)*

(define lista-propiedades

(list (cons prop1 1) (cons prop2 3) (cons prop3 6)

(cons prop4 8) (cons prop5 9) (cons prop6 11)

(cons prop7 13) (cons prop8 14)))

*; Tablero con propiedades*

*;; Función tablero-agregar-propiedad*

*;; Recorrido: tablero*

(define tablero-completo (tablero-agregar-propiedad tablero-vacio lista-propiedades))

;; Con lo definido en este script el tablero con las posiciones queda de la siguiente forma

;; (posicion entidad)

;; (0 salida) ;; casilla especial salida, todos comienzan en 0 luego a partir de acá se empieza a mover. Si al lanzar dados se terminan las posiciones se vuelve a posición 0 y se vuelve a contar. Ejemplo: jugador1 se encuentra en posicion 14 y la suma de dados es 2 entonces cae en posicion 1 (14 se mueve a 0, luego a 1)

;; (1 prop1) ;; casilla especial

;; (2 carcel) ;; casilla especial cárcel

;; (3 prop2)

;; (4 prop4)

;; (5 carcel) ;; casilla especial cárcel

;; (6 prop3)

;; (7 suerte) ;; casilla especial suerte

;; (8 prop4)

;; (9 prop5)

;; (10 comunidad) ;; casilla especial comunidad

;; (11 prop6)

;; (12 suerte) ;; casilla especial suerte

;; (13 prop7)

;; (14 prop8)

*; 5. Creación del juego*

*; Dominio TDA Juego = jugadores X tablero X dineroBanco X numeroDados X turnoActual (id jugador actual) X tasaImpuesto X maxCasas X maxHoteles*

*;; Anteriormente se tenia un último valor de "estadoJuego" donde se ejemplificaba con "en preparación". Sin embargo esto no tenía utilidad para el presente laboratorio por lo que se eliminó. Si usted lo implementa no hay problema pero tenga presente que no se utilizará en nada para este enunciado.*

(define g0 (juego '() tablero-completo 20000 2 0 10 4 1))

*; 6. Agregar jugadores al juego*

(define g1 (juego-agregar-jugador g0 p1))

(define g2 (juego-agregar-jugador g1 p2))

;; La ejecución de cada turno es llamar al requerimiento de juego-jugar-turno con el uso de la función (***juego-lanzar-dado*** semilla1 semilla2)

;; Valores de referencia de la semilla:

;;(getDadoRandom 1) ; retorna 1

;;(getDadoRandom 2) ; retorna 2

;;(getDadoRandom 5) ; retorna 3

;;(getDadoRandom 0) ; retorna 4

;;(getDadoRandom 3) ; retorna 5

;;(getDadoRandom 4) ; retorna 6

*;; En cada uno de sus scripts hasta el punto 6 es constante (sólo cambiará el número de jugadores, propiedades y cartas) pero puede usar el mismo esquema.*

z

*; 7. Jugar (inicio de simulación)*

(display "===== CAPITALIA =====\n\n")

;; Ambos jugadores comienzan en posición 0

*; Turno 1: Carlos*

**(display "TURNO 1: Carlos\n")**

**;; (juego-jugar-turno) Se destaca en amarillo para indicar que este comando es el que iniciará y ejecutará toda la simulación del juego. Es el único comando que ejecutarán para la ejecución de turnos y avance del juego.**

;; Parámetros:

;; g2: juego actual, cada nuevo juego se construye a partir del anterior  
;; (lanzar-dados 1 2) usa semilla 3 y 4

;; ***comprarPropiedad\_or\_construirCasa: #t***

;; ***construirHotel: #f***

;; ***pagarMultaSalirCarcel: #f***

;; ***usarTarjetaSalirCarcel: #f***

***(define g3 (juego-jugar-turno g2 (juego-lanzar-dados 1 2)******#t #f #f #f))***

*g3 ;; esto imprime g3 para que lo podamos evaluar.*

**;; Explicación:**

*;; Al usar la semilla 1 y 2 para dado1 y dado2, jugador Carlos se va a mover a posición 3 (0 + dado1:1 + dado2: 2)*

*;; En tablero posición 3 prop2:* Avenida Báltica *Precio 600*

*;; (define prop2 (propiedad 2 "Avenida Báltica" 600 4 #f 0 #f #f))*

;; Los últimos dos valores de juego-jugar-turno son para simplificar la simulación del juego

;; ***comprarPropiedad\_or\_construyeCasa:: #t*** para controlar la simulación se indica que se construya una casa si es que puede (si cae en una de sus propiedades y se dan las condiciones para que pueda construir 1 casa entonces se realiza, en caso de no cumplir las condiciones y el valor sea #t entonces se ignora la construcción de la casa. Internamente ustedes deben llamar a la función correspondiente dentro de (juego-jugar-turno)

;; ***construyeHotel: #f*** para controlar la simulación se agrega este campo para que construya un hotel en caso de que se cumplan las condiciones, al igual que el parametro de construyeCasa.

*;; dado que comprar casa en propiedad cuesta 600 y valor de* ***construyeCasa*** *es #t entonces su presupuesto queda en:*

*;; (1500-600) = 900*

*;; Dado que el parámetro de* ***construyeHotel*** *es #t pero no se cumplen las condiciones entonces se ignora.*

*;; g3 va a contener el valor del TDA juego actualizado*

*; Turno 2: Ana*

*;; Semilla 2 y 5 da: (dado1:2 dado2:3), jugador Ana se va a mover a posición 5, dado que se mueve de posición a 5 (2+3)*

***;;* En tablero: posición (5 cárcel) La posición 5 es cárcel por lo que pierde el turno**

**(display "TURNO 2: Ana\n")**

;; ***comprarPropiedad\_or\_construirCasa: #t***

;; ***construirHotel: #f***

;; ***pagarMultaSalirCarcel: #f***

;; ***usarTarjetaSalirCarcel: #f***

***(define g4 (juego-jugar-turno g3 (juego-lanzar-dados 2 5)******#t #f #f #f))***

*g4 ;; esto imprime g4*

*; Turno 3: Carlos. Carlos se encuentra en posición 3, al usar semilla 5 y 3 obtiene dado1: 3 y dado2: 5. Se mueve 8 posiciones, quedando en posición 11 (3 + 8)*

***;;* En tablero: posición (11 prop6) La posición 11 es prop6**

**;; Dinero actual jugador 1 Carlos = 900**

**;;** cre

;; ***comprarPropiedad\_or\_construirCasa: #t***

;; ***construirHotel: #f***

;; ***pagarMultaSalirCarcel: #f***

;; ***usarTarjetaSalirCarcel: #f***

***(define g5 (juego-jugar-turno g4 (juego-lanzar-dados 5 0)******#t #f #f #f))***

*g5 ;; esto imprime g5*

**;; Dinero actual jugador 1 Carlos = 900-800 (construyo casa) = 100**

**;; g5 entrega el estado actualizado del juego**

*; Turno 4: Ana. Ana se encuentra en posición 5 pero sigue en la cárcel. En este turno paga la multa para salir de la cárcel*

*; cómo paga multa (la multa es siempre 500) entonces su presupuesto queda: 1500-500 = 1000*

;; ***comprarPropiedad\_or\_construirCasa: #f***

;; ***construirHotel: #f***

;; ***pagarMultaSalirCarcel: #t***

;; ***usarTarjetaSalirCarcel: #f***

***(define g6 (juego-jugar-turno g5 (juego-lanzar-dado 3 4)******#f #f #t #f))***

*g6 ;; esto imprime g6*

*; Turno 5: Carlos.*

*;* **Dinero actual jugador 1 Carlos = 900**

**; Posición actual jugador 1 Carlos = 11**

**; usa semilla 1 y 2, dando los dados 1 y 2, por lo que la suma da 3**

**; movimiento es de 11 a 14 (11 + 3)**

**; Cae a posición 14 donde la propiedad cuesta 900 y cómo usa el valor #t de la propiedad entonces procede a comprar**

**: dado que llego a 0 entonces el jugador está en bancarrota y termina el juego**

**; el juego termina cuando uno de los jugadores llega a 0**

**; (define prop8 (propiedad 8 "Avenida Tennessee" 900 14 #f 0 #f #f))**

;; ***comprarPropiedad\_or\_construirCasa: #t***

;; ***construirHotel: #f***

;; ***pagarMultaSalirCarcel: #f***

;; ***usarTarjetaSalirCarcel: #f***

***(define g7 (juego-jugar-turno g6 (juego-lanzar-dado 1 2)******#t #f #f #f))***

*g7 ;; esto imprime g7*

*;; para verificar que todo esté correcto g7 va a entregar que el jugador 1 carlos tiene saldo/prespuesto 0 eso implica de que el jugador se encuentra en bancarrota y no se puede seguir jugando*

*; El comando jugador-esta-en-bancarrota se debe ejecutar al final del juego para el usuario vea en pantalla si efectivamente algun jugador se encuentra en bancarrota y que el juego termino*

**(jugador-esta-en-bancarrota jugador1) ;; va a entregar #t**

**;; Donde jugador1 lo pueden obtener con un selector del jugador en el juego**

**;; Por ejemplo pueden ejecutarlo como:**

**(jugador-esta-en-bancarrota (get-jugador g7))**

**;; Fin script ejecución**

1. https://docs.google.com/document/d/1cj6CsEN2ThY\_L2Wk880z7Y03Kk1vxpoqY9LVmUQPcuQ/edit?tab=t.0#heading=h.jiyysldll3vx [↑](#footnote-ref-0)