Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2025-2)

Tarea 1

Entrega

- **Fecha y hora oficial (sin atraso):** domingo 24 de agosto de 2025, 20:00.
- **Fecha y hora máxima (2 días de atraso):** martes 26 de agosto de 2025, 20:00.
- **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T1/`
  El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
- **Pauta de corrección**: **en este enlace**.
- **Formulario entrega atrasada: en este enlace**
- **Bases generales de tareas (descuentos)**: **en este enlace**.
- **Ejecución de tarea**: La tarea será ejecutada **únicamente** desde la terminal del computador. Además, durante el proceso de corrección, se cambiará el nombre de la carpeta "`T1/`" por otro nombre y se ubicará la terminal dentro de dicha carpeta antes de ejecutar la tarea. **Los *paths* relativos utilizados en la tarea deben ser coherentes con esta instrucción.**

Objetivos

- Desarrollar algoritmos para la resolución de problemas complejos.
- Aplicar competencias asimiladas en "Introducción a la Programación" para el desarrollo de una solución a un problema.
- Procesar *input* del usuario de forma robusta, manejando potenciales errores de formato.
- Interactuar con diversos archivos de Python y ejecutarlos mediante terminal.
- Trabajar con archivos de texto para leer, escribir y procesar datos.
- Escribir código utilizando paquetes externos (*i.e.* código no escrito por el estudiante), como por ejemplo, módulos que pertenecen a la biblioteca estándar de Python y módulos no estándares.
- Familiarizarse con el proceso de entrega de tareas y uso de buenas prácticas de programación.
- Aplicar conceptos de programación orientada a objetos (POO) para resolver un problema.

Índice

1. *DCCasillas*

En un rincón olvidado de la UC, entre pasillos polvorientos y pizarras llenas de ecuaciones, se habla de un misterioso juego llamado *DCCasillas*. Su origen es incierto: algunos dicen que nació como un simple pasatiempo entre ayudantes aburridos, otros aseguran que es una prueba secreta para entrenar a las mentes más brillantes en el arte de la programación y la lógica.

La leyenda cuenta que *DCCasillas* no es solo un juego, sino un desafío intelectual capaz de poner a prueba la paciencia, el ingenio y la habilidad de cualquier programador. El tablero parece inofensivo: una cuadrícula de números y casillas vacías, acompañada de metas claras en filas y columnas. Sin embargo, la verdadera dificultad está en encontrar la combinación perfecta de casillas.

Tal como en el mítico Estadio Zorros del Desierto, donde Cobreloa ha forjado victorias históricas bajo el sol implacable de Calama, aquí deberás mantener la mente fría y la estrategia afilada. Un error puede costarte el partido, pero una jugada perfecta puede llevarte directo a la gloria …y al podio del ranking.

Tu misión, como valiente estudiante de Programación Avanzada, es convertirte en el arquitecto de este rompecabezas digital. Deberás leer y procesar configuraciones, manipular casillas, verificar soluciones y, cuando sea necesario, idear un algoritmo capaz de resolver automáticamente incluso los tableros más rebeldes.

¡Pero cuidado! el sistema es implacable. Cada movimiento será registrado, cada error castigado, y no habrá salvación si tu código no cumple con las reglas establecidas. El destino de tu puntaje (y de tu nota) dependerá de tu precisión y creatividad.

Prepárate. *DCCasillas* te espera.

2. Juego

DCCasillas es un juego individual que consiste en un conjunto de T tableros que el jugador debe resolver.

Cada tablero es una cuadrícula rectangular de $N \times M$ casillas que llamaremos **casillas regulares**. Una casilla regular puede contener un número entero positivo o estar vacía. Una casilla regular que contiene un número puede estar habilitada o deshabilitada. Todas las casillas regulares están inicialmente habilitadas. Por ejemplo, un tablero de 3×5 puede ser la siguiente.

2		1		4
3		2		1
4	2	3	2	2

Figura 1: Tablero inicial de 3×5 casillas regulares, todas habilitadas

Además de las $N \times M$ casillas regulares, en el extremo derecho y en el extremo inferior, se agregan una columna y fila adicional, que contienen **casillas objetivo**. Una casilla objetivo puede estar vacía o contener un entero positivo que indica el resultado que se espera obtener sumando las casillas regulares habilitadas de la fila o columna correspondiente. La casilla de la esquina inferior derecha siempre está vacía.

Por ejemplo, al agregar las casillas objetivo al tablero de la Figura 1, este quedará como en la Figura 2.

El objetivo del juego es deshabilitar las casillas regulares que sean necesarias para que, en cada fila y columna, la suma de sus casillas habilitadas corresponda al valor indicado en la correspondiente casilla objetivo. Si una casilla objetivo está vacía, entonces no se considera la suma de la fila o columna correspondiente.

2		1		4	
3		2		1	
4	2	3	2	2	10
7		3		3	

Figura 2: Tablero inicial de 3×5 casillas regulares, y 9 casillas objetivo. Las casillas objetivo se muestran en gris.

Por ejemplo, en la tabla de la Figura 2, el objetivo de la tercera fila es que las celdas habilitadas sumen 10 (en la configuración inicial suman 13). En cuanto a columnas, la primera columna debe sumar 7 (inicialmente suma 9), la tercera columna debe sumar 3 (inicialmente suma 6) y la quinta columna debe sumar 3 (inicialmente suma 7). Las otras filas y columnas tienen una celda objetivo vacía.

Durante el juego, el jugador puede deshabilitar y rehabilitar casillas regulares. Una casilla regular deshabilitada no se suma en su fila o columna correspondiente. Una casilla regular deshabilitada puede volver a ser habilitada, en cuyo caso recupera el valor entero que tenía inicialmente. Las casillas objetivo no pueden ser modificadas.

Una posible configuración que soluciona el problema que hemos usado de ejemplo hasta ahora, en que se han deshabilitado 3 casillas, es la representada en la Figura 3.

■		1		■	
3		2		1	
4	2	■	2	2	10
7		3		3	

Figura 3: Tablero resuelto de 3×5 con 3 casillas deshabilitadas.

Un tablero resuelto tiene un puntaje asociado. El puntaje corresponde a la cantidad de movimientos: habilitar y deshabilitar casillas, que el jugador realizó para llegar a la solución. El puntaje total del juego es la suma de los puntajes obtenidos en todos los tableros resueltos.

En esta tarea deberás implementar una versión de este juego que permita leer un conjunto inicial de tableros, deshabilitar y rehabilitar casillas regulares en cada uno de ellos, y verificar la validez de una posible solución. Adicionalmente, deberás poder almacenar un estado del juego y posteriormente recuperarlo y almacenar estados parciales. Finalmente, deberás programar un algoritmo que permita encontrar una solución válida a partir de la configuración inicial, o bien determinar que no existe una solución al tablero.

3. Flujo del programa

Para esta tarea se deben completar dos partes:

Parte 1 Completar el modelamiento de las clases entregadas, llamadas `DCCasillas` y `Tablero`. Estas clases modelarán respectivamente el juego completo, y un tablero particular. La clase `DCCasillas` debe mantener una lista de `Tableros`, los cuales deben poder ser leídos y escritos desde/hacia archivos, además de mantener el estado del usuario acutal. Por otro lado, la clase `Tablero` debe permitir manipular los estados de las casillas del tablero, determinar si el estado del tablero es una solución, determinar si una solución existe y resolver el tablero automáticamente.
Esta parte será corregida automáticamente mediante el uso de *tests*.

Parte 2 Construir un menú por consola, que permita que el jugador interactúe con *DCCasillas*.
Esta parte será corregida manualmente por el cuerpo docente.

3.1. Sobre los *tests*

Junto con el código base, se entregará un conjunto de *tests* que les permitirán verificar distintos casos en base a su respuesta esperada. Este conjunto de *tests* serán los "***tests* públicos**". Los *tests* públicos evalúan distintos casos de uso del programa, pero no necesariamente cubren todos los casos posibles. En la etapa de corrección, el equipo docente usará un conjunto de *tests* distintos que serán los ***tests* privados** y contienen *tests* que pueden evaluar *inputs* distintos y casos que pueden no haber sido incluidos en los *públicos*. Posteriormente a la entrega, se publicarán también los *tests* privados de manera que puedan corroborar su puntaje.

Por lo tanto, **es responsabilidad del estudiantado confeccionar una solución que cumplan con lo expuesto en el enunciado y que no esté creada solamente a partir de los *tests* públicos**. De ser necesario, el estudiantado deberá pensar en nuevos casos que sean distintos a los *tests* públicos. **No se aceptarán supuestos que funcionaron en los *tests* públicos, pero que van en contra de lo expuesto en el enunciado.**

Para cada clase y método indicado en el enunciado, **no puedes modificar su nombre, agregar nuevos argumentos o argumentos por defecto**. Tampoco está permitido cambiar el nombre y ubicación de los archivos entregados. En caso de no respetar esto, se aplicará un descuento importante. Puedes crear nuevos atributos, nuevos métodos, archivos y/o funciones si estimas conveniente. También se permite crear funciones en otro módulo y que el método que pedimos completar únicamente llame a esa función externa. Sin embargo, para que los *tests* funcionen correctamente, los métodos y atributos explícitamente indicados aquí deben respetarse.

3.2. Parte 1 - Funcionalidades

En esta parte se debe completar el modelamiento de las clases `DCCasillas` y `Tablero`, para las cuales se entrega un código base inicial. Para todos los métodos que impliquen abrir un archivo del directorio `config`, puedes suponer que el formato de los archivos será correcto.

Modificar `class Tablero:`
Clase que representa un tablero del juego *DCCasillas*.

- Modificar `def __init__(self) -> None:`
  Inicializa una instancia de tablero. Inicializa al menos los siguientes atributos:

`self.tablero`	Una `list` de `list`s que contiene las casillas del tablero.
`self.movimientos`	La cantidad de acciones que se han aplicado al tablero hasta llegar a su estado actual. Siempre empieza en 0.
`self.estado`	Un `bool` que indica si el tablero se encuentra resuelto (`True`) o no (`False`).

  No puede modificar el nombre ni la cantidad de parámetros del método, pero sí podría desear agregar atributos adicionales si lo estima conveniente.

- Modificar `def cargar_tablero(self, archivo: str) -> None:`
  Recibe un nombre de archivo con el contenido del tablero. El archivo debe encontrarse en el directorio `config/`. Debes abrir el archivo, usando el *encoding* `utf-8`, leer y procesar su contenido, y construir el contenido del atributo `self.tablero`. El formato del archivo se describe en Formato de entrada y salida. Puedes suponer que el nombre del archivo entregado existe dentro de la carpeta `config/`, que `archivo` incluye la extensión `.txt`, y que el archivo está correctamente construido.

- Modificar `def mostrar_tablero(self) -> None`
  Este método utiliza el módulo `visualizador` para mostrar en pantalla el contenido del tablero. Su uso está descrito en `visualizador.pyc`.

- Modificar `def modificar_casilla(self, fila:int, columna:int) -> bool`
  Modifica el estado de la casilla indicada por `fila` y `columna`. Si la casilla estaba habilitada, debe quedar deshabilitada, y viceversa. Debe verificar que los parámetros hagan referencia a una casilla regular válida. Este método debe retornar `True` en caso que la acción haya podido realizarse, y `False` en caso contrario, por ejemplo, si los parámetros de entrada son incorrectos.

- Modificar `def validar(self) -> bool`

  Verifica si la configuración del tablero es una solución válida de acuerdo a las reglas del juego. Retorna `True` si lo es, o `False` en caso contrario. De ser necesario debe actualizar el estado del tablero.

- Modificar `def encontrar_solucion(self) -> Tablero`

  Retorna una instancia de `Tablero` que representa una solución para el `Tablero` actual, **sin modificar el `Tablero` actual**. Si el tablero actual ya es una solución, entonces retorna una **copia** del mismo tablero actual. Si el tablero no tiene solución debe retornar `None`.

  Hint: Si bien existen diversas maneras de encontrar de solucionar un tablero, o bien de determinar que el tablero no es válido, la técnica de *backtracking* puede ser particularmente útil en este caso.

Modificar `class DCCasillas:`
Clase que representa el juego. Al crear un juego, se carga una serie de tableros. El jugador puede abordar los tableros en cualquier orden.

- Modificar `def __init__(self, usuario: str, config: str) -> None:`

  Inicializa una instancia del juego. El juego debe inicializar al menos los siguientes atributos.

`self.usuario`	Un `str` con el usuario actual. No puede ser vacío.
`self.puntaje`	Un `int` con el puntaje actual del usuario, inicializado en 0.
`self.tablero_actual`	Un `int` que indica el número del `Tablero` que está siendo jugado actualmente. Inicialmente `None`.
`self.tableros`	Una `list` de `Tableros` que contiene todos los tableros disponibles en el juego. Se inicializa a partir del archivo `config`, y cuyo formato se describe en Formato de entrada y salida.

- Modificar `def abrir_tablero(self, num_tablero: int) -> None:`
  Cambia el `Tablero` actual al especificado por el índice `num_tablero`.

- Modificar `def guardar_estado(self) -> bool:`
  Guarda el estado del juego en un archivo con el nombre del usuario actual, dentro del directorio `data`. El formato está indicado en Formato de entrada y salida. Retorna `True` en caso que el estado haya podido guardarse correctamente, y `False` en caso contrario. Si ya había una archivo con ese nombre, se sobreescribe.

- Modificar `def recuperar_estado(self) -> bool:`
  Recupera el estado del juego a partir del archivo indicado por el nombre de usuario y la extensión `.txt`. El formato está descrito en Formato de entrada y salida. En caso que el estado haya podido recuperarse correctamente, retorna `True`. En caso contrario retorna `False`.

3.3. Parte 2 - Menú

En esta parte deberás implementar un menú que permita interactuar con el juego *DCCasillas*. Para esto, se ejecutará tu programa mediante un archivo `main.py`. Las acciones a ejecutar en los menús deberán aceptar entradas de usuario mediante terminal.

Se deben crear al menos dos menúes: Menú de Juego y Menú de Acciones. En esta parte de la tarea, se evaluará principalmente que (1) ambos menús sean a prueba de errores, (2) ambos incluyan al menos todas las opciones solicitadas, (3) que los parámetros recibidos por el usuario se manejen de forma correcta, y (4) que cada opción del menú llame, mediante código, a la función o método correspondiente.

En esta parte no se evaluará si la opción seleccionada realmente hace lo esperado, ya que eso será evaluado en la Parte 1 mediante *tests*. En esta parte se espera un correcto manejo de *inputs* y llamar, en el código, a las funciones o métodos que correspondan según la opción elegida.

3.3.1. Menú de Juego

Este es el menú encargado de iniciar la interacción con el usuario y realizar la carga de *DCCasillas*. A continuación se entrega un ejemplo de cómo se podría ver el menú de inicio tras haber ejecutado por primera vez `python3 main.py` :

```
¡Bienvenido a DCCasillas!

Usuario: UUU, Puntaje: PPP
Tableros resueltos: XXX de YYY

*** Menú de Juego ***

[1] Iniciar juego nuevo
[2] Continuar juego
[3] Guardar estado de juego
[4] Recuperar estado de juego
[5] Salir del programa

Indique su opción (1, 2, 3, 4, 5):
```

Figura 4: Ejemplo de Menú de Inicio

El menú inicio debe mostrar: el usuario actual del juego; el puntaje actual del usuario; la cantidad de tableros del juego actual y la cantidad de tableros resueltos. Al entrar por primera vez al menú estos valores están vacíos.

Las acciones del Menú de Juego deben ser las siguientes::

1. **Iniciar juego nuevo**: Empieza un nuevo juego. Si no hay nombre de usuario definido, debe solicitar un nombre e iniciar un nuevo juego con este nombre de usuario, puntaje 0, y con una configuración de tablero que debe cargar desde un archivo cuya descripción se encuentra en Subsubsección 3.4.3. Si la configuración de tablero existe, debe pasar al Menú de Acciones y usar el tablero 0 como actual. En caso contrario, se debe comunicar el error y mantenerse en este menú.

   Si ya había un nombre de usuario definido, se debe dar la opción de cambiarlo. En cualquier caso, esta opción de menú debe iniciar un nuevo juego con puntaje 0 y ningún tablero resuelto.

2. **Continuar juego**: Debe dar la opción de elegir un nuevo número de tablero o continuar con el actual. Si se elige un tablero válido, debe pasar al Menú de Acciones y continuar con el tablero. Si no hay un usuario definido, debe comunicar el error y mantenerse en este menú.

3. **Guardar estado de juego**: Guarda el estado actual del juego en un archivo dentro de la carpeta `data/` con el nombre del usuario actual y extensión `.txt`. Si no hay un nombre de usuario definido esta acción no puede realizarse y se debe comunicar el error. En cualquier caso debe mantenerse en el menú actual.

El formato del archivo de guardado está descrito en Formato de guardado de tablero.

4. **Recuperar estado de juego**: Carga el estado de un juego desde un archivo dentro de la carpeta `data/` con el nombre del usuario actual y extensión `.txt`. Cargar un juego implica actualizar todos los tableros y estado desde el archivo, sobrescribiendo el estado actual. Luego de la carga debe mantenerse en el menú actual. Si la carga no pudo realizar (por ejemplo, porque no existe el archivo), debe comunicar el error.
5. **Salir del programa**: Termina la ejecución del programa. No guarda ningún estado del juego actual.

3.3.2. Menú de Acciones

Luego de haber seleccionado exitosamente la opción 1 ó 2 del Menú de Juego, se abrirá un segundo menú encargado de realizar las acciones sobre los tableros. Se debe indicar cuál es el tablero actual, y ofrecer las siguientes acciones:

```
DCCasillas
Usuario: UUU, Puntaje: PPP
Número de tablero: XXX de YYY
Movimientos tablero: ppp


*** Menú de Acciones ***

[1] Mostrar tablero
[2] Habilitar/deshabilitar casillas
[3] Verificar solución
[4] Encontrar solución
[5] Volver al menú de Juego

Indique su opción (1, 2, 3, 4, 5):
```

Figura 5: Ejemplo de Menú de Acciones

Las acciones del menú de acciones deben ser las siguientes:

1. **Mostrar tablero**: Despliega en pantalla el estado actual del tablero Para esto debe usar la función `imprimir_tablero`, disponible en el archivo precompilado `visualizador.pyc`.
2. **Habilitar/deshabilitar casillas**: Cambia el estado de una casilla. Si se intenta ejecutar la acción sobre un casilla no permitida, debe indicar el error y continuar. Cada cambio de casilla suma un movimiento al puntaje del usuario, el cual debe reflejarse en el menu de acciones la próxima vez que se muestre. Luego de ejecutar la acción debe mostrar el nuevo estado del tablero.
3. **Verificar solución**: Revisa si el estado actual del tablero es o no una solución. Si la solución es correcta, el usuario suma como puntaje la cantidad de movimientos habilitar/deshabilitar que se han ejecutado sobre el tablero. Si la solución no es correcta, no se suma puntaje.
4. **Encontrar solución**: Calcula una solución a partir del estado actual del tablero. Si existe una solución, debe mostrarla usando la opción de mostrar el tablero. Se agrega al puntaje del usuario una cantidad de puntos equivalente a la cantidad de casillas regulares del tablero.
5. **Volver al menú de juego**: Regresa al menú de juego.

3.4. Formato de entrada y salida

Aquí se describen los formatos de los archivos de entrada y salida del programa. Todos los archivos deben considerarse como archivos de texto en codificación `UTF-8`.

3.4.1. Formato de entrada de tablero

Para describir un tablero regular, la primera línea contiene dos enteros `N` y `M` separados por un espacio, que indican respectivamente la cantidad de filas y cantidad de columnas de las casillas regulares.

A continuación se encuentran `N+1` líneas. Las primeras `N` líneas corresponden a una fila del tablero. Dentro de cada fila hay `M+1` posiciones separadas por un espacio. Cada posición puede contener a un entero o bien al caracter `'.'`. Las primeras `M` posiciones corresponden a una casilla regular, y la última posición corresponde a la casilla objetivo de esa fila. La última fila del archivo también contiene `M+1` posiciones separadas por un espacio, que también pueden contener a un entero o bien al caracter `'.'`. Cada una de esta casillas corresponde a una casilla objetivo de la columna correspondiente. La última posición de esa última fila siempre será un `'.'`.

Por ejemplo, para el tablero de ejemplo de la izquierda, el archivo de entrada correspondiente se muestra a la derecha:

2		1		4	
3		2		1	
4	2	3	2	2	10
7		3		3	

```
3 5
2 . 1 . 4 .
3 . 2 . 1 .
4 2 3 2 2 10
7 . 3 . 3 .
```

Figura 6: Tablero inicial y su correspondiente archivo de entrada

3.4.2. Formato de guardado de tablero

Para guardar el estado de un juego, se utiliza un archivo que tenga el nombre del jugador actual y extensión `.txt`. En la primera línea hay un entero T con la cantidad de tableros descritos. A continuación, por cada tablero hay una secuencia donde la primera línea contiene un entero con la cantidad de movimientos efectuados sobre el tablero, luego una línea con los enteros `N` y `M` que corresponden a la cantidad de filas y columnas de casillas regulares, y finalmente `N+1` líneas la configuración almacenada como se describe en Formato de entrada de tablero. Las celdas deshabilitadas se almacenan con una `X` antes del entero correspondiente.

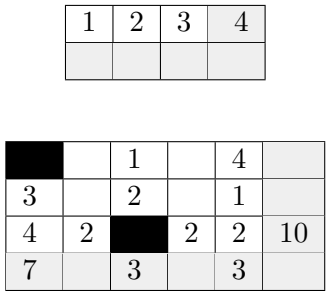
Por ejemplo, en la configuración de la Figura 7 hay dos tableros. El primer tablero no registra movimientos. En el segundo tablero, el jugador ha realizado 6 movimientos. En ese tablero se han deshabilitado dos casillas, como se muestra a la izquierda. A la derecha se muestra el contenido del archivo.

Este formato también se usa como entrada para recuperar el estado de un juego.

3.4.3. Formato de configuración de juego

Para describir la secuencia de tableros de un juego, se utiliza un archivo de texto. En la primera línea hay un entero positivo `T` con la cantidad de tableros que componen este juego. A continuación hay `T` líneas, cada una con un nombre de archivo que contiene la descripción de un tablero. No hay ninguna restricción sobre el nombre o extensión de los archivos. Tanto el archivo de configuración como los archivos de tableros deben encontrarse en el directorio `config`.

Por ejemplo, un archivo de configuración posible con 5 tableros es el siguiente:



1	2	3	4

		1		4	
3		2		1	
4	2		2	2	10
7		3		3	

```
2
0
1 3
1 2 3 4
. . . .
6
3 5
X2 . 1 . 4 .
3 . 2 . 1 .
4 2 X3 2 2 10
7 . 3 . 3 .
```

Figura 7: Archivo de guardado para 2 tableros

```
5
tablero01.txt
tablero02.txt
tablero03.txt
tablero04.txt
tablero05.txt
```

Figura 8: Ejemplo de archivo de configuración `config.txt` con 5 tableros.

4. Archivos

Aquí describe la estructura base del código de la tarea, y los archivos relevantes que deben estar en su repositorio.

4.1. Código inicial

Se te entregarán algunos archivos que deberás completar. Puedes crear más archivos si lo estimas conveniente.

- Modificar `dccasillas.py`: Contiene la definición de la clase `DCCasillas`.
- Modificar `tablero.py`: Contiene la definición de la clase `Tablero`.
- Crear `main.py`: Archivo principal que ejecuta el menú por la consola.
- No modificar `visualizador.pyc`: Este archivo contiene la función `imprimir_tablero` que deberás para mostrar un tablero en la consola. Su uso se describe en la sección `visualizador.pyc`.
- No modificar `data/`: Esta carpeta contendrá archivos de extensión `.txt` que almacenan estados del juego. Estos archivos usan el formato indicado en 3.4.2. Cada usuario puede tener solo un archivo de estado (o ninguno).
- No modificar `config/`: Esta carpeta contendrá archivos de configuración y archivos de entrada de tableros. Estos archivos están descritos en 3.4.1 y 3.4.3.
- No modificar `tests_publicos/`: Esta carpeta contendrá una serie de archivos `.py` que corresponden a diferentes *tests* para apoyar el desarrollo de la Parte 1. Puedes utilizar los archivos entregados para ir viendo si lo desarrollado hasta el momento cumple con lo esperado en esta evaluación.

Importante: los *tests* entregados en esta carpeta no serán los mismos que se utilizarán para la corrección de la evaluación.

- Modificar `README.md`: Contendrá inicialmente las actualizaciones a la tarea aplicadas hasta el momento. Debe modificarse para personalizar tu propio `README`.
- No modificar `README_inicial.md`: Es la base desde la cual deberás construir tu propio `README`.

4.2. `visualizador.pyc`

Para facilitar tu trabajo y evaluar el uso de módulos, se te hará entrega del módulo visualizador.pyc que contiene la función `imprimir_tablero` que deberás utilizar en la tarea. Este archivo se encuentra compilado, por lo que no se puede visualizar su contenido, y funciona únicamente con la versión de Python del curso: 3.12.X. Utilizar una versión distinta a 3.12 provocará un error de **magic number**. Este archivo se importa como cualquier módulo, por ejemplo con `import visualizador`.

La función que contiene es: `imprimir_tablero(tablero: list) -> None`: Recibe una `list` de `lists` con el formato del atributo `tablero` de la clase `Tablero`. Esta función muestra en la consola una visualización del tablero con los valores de cada celda, y muestra las celdas vacías con un espacio en blanco.

La función espera que todas las celdas de tablero sean un `str` y que las casillas vacías contengan el caracter `'.'`. La manera en que se representan las celdas habilitadas queda a criterio del programador, pero al momento de entregar el parámetro a esta función, deben ser `str`s.

Debes importar correctamente este archivo y hacer uso de su función para el desarrollo de la tarea.

5. `.gitignore`

Para esta tarea **deberás utilizar un** `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T1/`.

Los elementos que no debes subir y **debes ignorar mediante** `.gitignore` para esta tarea son:

- El enunciado.
- El archivo `visualizador.pyc`
- La carpeta `config/` y todo lo contenido en ella.
- La carpeta `data/` y todo lo contenido en ella.
- La carpeta `tests_publicos/` y todo lo contenido en ella.
- El archivo `README_inicial.md`.

Recuerda **no ignorar archivos vitales de tu tarea como los que tú debes modificar, o tu tarea no podrá ser revisada**. Es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos **deben** no subirse al repositorio debido al uso correcto del archivo `.gitignore` y no debido a otros medios.

6. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, **sólo se corrigen tareas que se puedan ejecutar**. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios.

En el **siguiente enlace** se encuentra la distribución de puntajes. En color **amarillo** se encuentra cada ítem que será evaluado a nivel funcional y de código, es decir, aparte de que funcione, se revisará que el código esté bien confeccionado y que la funcionalidad esté correctamente integrada en el programa. Todo aquel que no esté pintado de amarillo significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea o con los *tests*.

Importante: Todo ítem corregido por el cuerpo docente será evaluado únicamente de forma ternaria: cumple totalmente el ítem, cumple parcialmente o no cumple con lo mínimo esperado. Del mismo modo, todo ítem corregido automáticamente será evaluado de forma ternaria: puntaje completo si pasa todos los *tests* de dicho ítem, medio punto para quienes pasan más del 70 % de los *test* de dicho ítem, y 0 puntos para quienes no superan el 70 % de los *tests* en dicho ítem. Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente respetando lo expuesto en **el documento de bases generales**.

La corrección se realizará en función del último *commit* realizado antes de la fecha oficial de entrega. Si se desea continuar con la evaluación en el periodo de entrega atrasado, es decir, realizar un nuevo *commit* después de la fecha de entrega, **es imperante responder el formulario de entrega atrasada** sin importar si se utilizará o no cupones. Responder este formulario es el mecanismo que el curso dispone para identificar las entregas atrasadas. El enlace al formulario está en la primera hoja de este enunciado.

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

6.1. Ejecución de *tests*

Para la corrección automática se entregarán varios archivos `.py` los cuales contienen diferentes *tests* que ayudan a validar el desarrollo de la Parte 1 - Funcionalidades. Para ejecutar estos *tests*, primero debes posicionar tu terminal/consola en la carpeta de la tarea `Tareas/T1/`. Luego, desde esta misma, debes escribir el siguiente comando para ejecutar todos los *tests*:

- `python3 -m unittest discover tests_publicos -v -b`

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes hacerlo escribiendo lo siguiente:

- `python3 -m unittest -v -b tests_publicos.<test_N>`
  Reemplazando `<test_N>` por el test que desees probar.

Por ejemplo, si quisieras probar si realizaste correctamente el método `modificar_casilla` de `Tablero`, deberás escribir lo siguiente:

- `python3 -m unittest -v -b tests_publicos.test_00_modificar_casilla`

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.12`.

7. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.12.X con X mayor o igual a 0.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta. **No se revisará archivos en otra extensión como `.ipynb`**.

- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python **está prohibido**. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo o bien incluirlo pero que se encuentre vacío conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a la publicación del enunciado. Además, se encuentra estrictamente prohibido el uso de herramientas generadoras de código para el apoyo de la evaluación.
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0)**.**