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

IIC2233 Programación Avanzada (2023-2)

Tarea 1

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso):** miércoles 6 de septiembre de 2023, 20:00
  - **Fecha y hora máxima (2 días de atraso):** viernes 8 de septiembre de 2023, 20:00.
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T1/`

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.
- 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.
- 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.
- Utilizar *properties* como herramientas de modelación.

Índice

1. *DCChexxploding*

Durante las vacaciones, **Gatochico** decidió aprender a jugar Ajedrez, por lo que pasó unas entretenidas tardes aprendiendo y jugando en internet este grandioso juego. Una vez que aprendió todo lo que debía saber, se decidió a probar algo nuevo. Es por esto, que se propuso inventar un nuevo juego llamado *DCChexxploding*. En este juego, habrán piezas explosivas que destruirán el tablero, y es trabajo del jugador posicionar diferentes peones para que las piezas explosivas no destruyan todo el tablero, y haya una destrucción controlada.

Ahora, te pide a ti, como estudiante de *Programación Avanzada* que desarrolles un programa para resolver diferentes tableros de *DCChexxploding*.

2. Reglas del sistema de defensa

Para poder desarrollar *DCChexxploding*, deberás simular las secciones del tablero. Este tablero tiene forma de un tablero rectangular, con `n` filas y `m` columnas. Cada una de las celdas puede estar en tres posibles estados:

- `"--"`: Representa una **celda vacía**.
- `"PP"`: Representa una **peón** en dicha posición.
- `"XN"`: Representa una **pieza explosiva**. El primer carácter, `"X"` será una de las siguientes 3 letras: `"V"`, `"H"`, `"R"`. Mientras que el segundo carácter, `"N"` corresponderá a un número entero mayor a 0 (`int`), el cual indica la cantidad de celdas que la pieza debe ser capaz de destruir contándose a sí misma. Por ejemplo, `"V4"` es una pieza explosiva vertical que debe destruir a 4 celdas del tablero. En la siguiente hoja se explica cada tipo de pieza y cómo afecta a la explosión.

Por temas de estandarización, para un tablero de medidas `MxN` el origen del tablero estará ubicada en la esquina superior izquierda, siendo esta la coordenada `(0, 0)`. En el lado opuesto, la esquina inferior derecha será la última celda ubicada en la coordenada `(N - 1, M - 1)`. El primer número corresponde a la fila del tablero, mientras que el segundo número corresponde a la columna. A continuación se muestra un ejemplo de un tablero de `4x5`:

	0	1	2	3	4	5
0	V2		PP			
1		PP				
2	PP		R5	PP		PP
3			PP			
4	H3			PP		

Figura 1: Tablero de *DCChexxploding* de `4x5`

Tal como se muestra en el ejemplo, en la coordenada `(0, 0)` está ubicada una pieza del tipo `"V"` con el número `2`. En la coordenada `(2, 5)` está la celda con un peón (`"PP"`), la coordenada `(3, 4)` está la celda vacía (`"--"`) y en la coordenada `(4, 0)` está la pieza del tipo `"H"` con el número `3`.

Adicionalmente, se define el **alcance de una pieza explosiva** como la cantidad de celdas que son afectadas por la explosión de una pieza explosiva. Las celdas afectadas por la explosión son la misma celda donde está la pieza explosiva y, según el tipo de pieza, toda aquella que comparta fila, columna o diagonal con la pieza explosiva. Además, no debe existir ningún peón entre la celda y la pieza explosiva. La explosión de esta pieza dependerá de su tipo:

- Tipo `"V"`: Pieza explosiva **vertical**. Solo afecta a celdas que estén en la misma columna que la pieza explosiva.
- Tipo `"H"`: Pieza explosiva **horizontal**. Solo afecta a celdas que estén en la misma fila que la pieza explosiva.
- Tipo `"R"`: Pieza explosiva **reina**. Solo afecta a celdas que estén en la misma fila, columna o en diagonal a esta pieza.

Por ejemplo, en la Figura 2 se destacó cada celda que está dentro del alcance de una pieza explosiva **reina**. En el tablero de la izquierda en donde no hay peones, el alcance de esta pieza es 13. En cambio, en el tablero de la derecha, se incluyeron peones que se encuentran entre algunas celdas que antes eran afectadas y la pieza explosiva. Por lo tanto, ahora su alcance es 7.

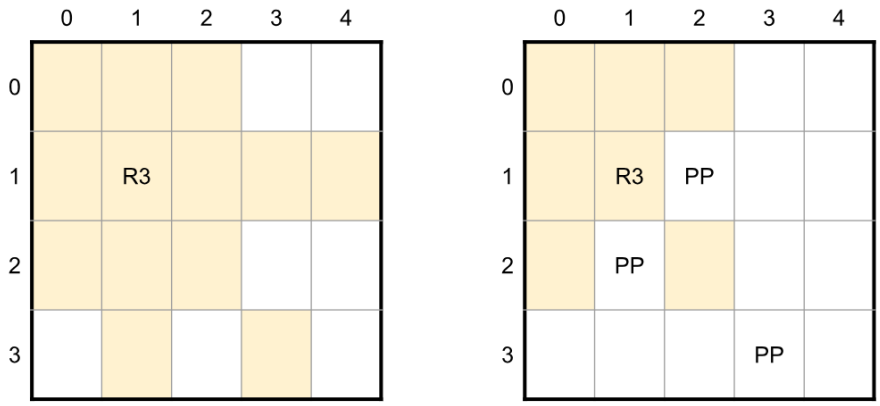

Figura 2: Alcance de una pieza explosiva **reina**. En el tablero de la izquierda su alcance es 13 y en el de la derecha es 7.

La forma de rellenar las celdas con peones tiene una variedad de reglas que permite que puedan existir una o más soluciones. Las reglas que se deben cumplir son las siguientes:

Regla 1 Cada número de una pieza explosiva representa la cantidad **exacta** de celdas que deben ser afectadas por la explosión de una pieza.

Un tablero de *DCChexxploding* cumple esta regla cuando todas las piezas explosivas del tablero cumplen esta regla, es decir, el alcance de cada pieza explosiva es igual al número informado en esta.

Por ejemplo, si revisamos el caso de la Figura 1, podemos notar que en la coordenada `(0, 0)`, la pieza es del tipo `"V"`, es decir, solo afecta a las celdas verticales. Y dado que tiene el número `2`, significa que su explosión debe afectar a `2` celdas verticales incluyendo la celda en donde está instalada la pieza. Como hay un peón en `(2, 0)`, efectivamente el alcance de esta pieza es igual al número informado porque la explosión solo alcanza a la celda `(0, 0)` y `(1, 0)`. Por lo tanto, esta pieza cumple la regla.

Regla 2 Las piezas explosivas no pueden ser tapadas por peones, es decir, cuando se soluciona el tablero, no se puede posicionar un peón sobre una pieza explosiva en la misma celda.

Regla 3 Se define un "vecino" como la celda que está directamente al lado, horizontal o verticalmente, y no en diagonal. La última regla es que un peón solo puede tener máximo a un vecino que sea peón. Las demás celdas deben ser vacías o tener piezas explosivas.

Un tablero de *DCChexxploding* se considera resuelto cuando cumple las 3 reglas. A continuación se muestra un ejemplo del tablero original con dos posibles soluciones.

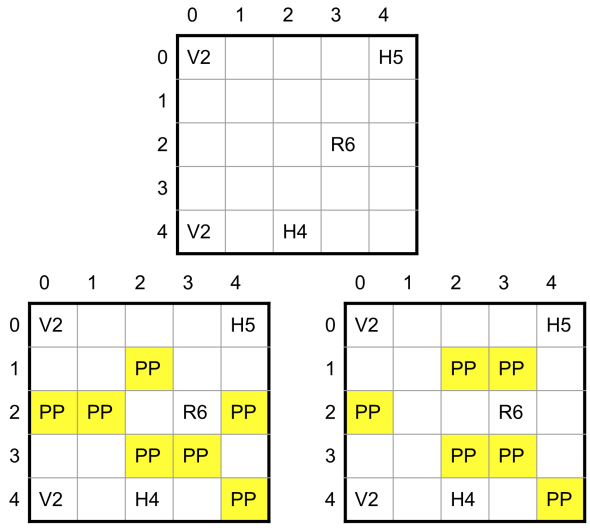

Figura 3: Tablero de *DCChexxploding* y dos posibles soluciones. Se pintó en amarillo a los peones solo con fines de destacar las celdas modificadas para solucionar el tablero.

3. Flujo del programa

Tu objetivo en esta evaluación estará dividido en completar 2 partes:

Parte 1 Completar 2 archivos con la clase `PiezaExplosiva` y `Tablero`, los cuales permitirán: abrir, analizar y solucionar diferentes tableros de *DCChexxploding*. Para lograr lo anterior, se te entregará un conjunto de archivos y *tests* que validarán la correcta implementación de ambas clases.
Esta parte será corregida automáticamente mediante el uso de *tests*.

Parte 2 Confeccionar un menú por consola, que permita interactuar con un tablero de *DCChexxploding*.
Esta parte será corregida manualmente por el cuerpo docente.

3.1. Parte 1 - Funcionalidades (70 %)

En esta parte, deberás completar diversos métodos y *properties* de las clases indicadas para trabajar con el tablero de *DCChexxploding*. La corrección completa de esta partes será mediante *tests*. Es por esto que debes asegurarte que cada método y *property* se pueda ejecutar correctamente, y que el archivo no presente errores de sintáxis.

Para cada clase, método y *property* indicada a continuación, **no puedes modificar su nombre, agregar nuevos argumentos o argumentos por defecto**. Puedes crear nuevos métodos, archivos y/o funciones si estimas conveniente, pero debes mantener el formato de los métodos entregados. En caso de no respetar lo indicado, la evaluación presentará un fuerte descuento.

A continuación se describen las 2 clases que deberás completar: `PiezaExplosiva` y `Tablero`.

Modificar `class PiezaExplosiva`:
Clase que representa una pieza explosiva del Tablero. Registra la cantidad de celdas que debe destruir esta pieza, tipo de pieza, y su posición en el tablero.

- **No modificar** `def __init__(self, alcance: int, tipo: str, posicion: list) -> None`:
  Inicializa una instancia de `PiezaExplosiva` y guarda como atributo el alcance de la explosión, su tipo y su posición.

- **No modificar** `def __str__(self) -> str`:
  Retorna un texto con la descripción de la pieza.

- **Modificar** `def verificar_alcance(self, fila: int, columna: int) -> bool`:
  Verifica si la posición indicada por `(fila, columna)` es una posición alcanzable por la explosión de la pieza. Puedes asumir que `fila` y `columna` serán siempre números iguales o mayores a 0.

  Ejemplo: si la pieza actual es del tipo vertical (`V`), toda posición que comparta la misma columna con la pieza, estará en una posición alcanzable a la explosión, y se retornará `True`. En otro caso, se retornará `False`.

Modificar `class Tablero`:
Clase que representa un Tablero de *DCChexxploding*. Esta clase registra el tablero como una lista de listas de *strings*, y las dimensiones de dicho tablero (cantidad de filas y columnas). También permite efectuar diversos métodos y *properties* para interactuar con el tablero.

- **No modificar** `def __init__(self, tablero: list) -> None`:
  Inicializa una instancia de `Tablero` y guarda, como atributo, el tablero y sus dimensiones. El atributo `tablero` siempre corresponderá a una lista de listas de *strings*.

- **Modificar** `def desglose(self) -> list`:
  Property que utiliza el tablero guardado como atributo y retorna una lista de 3 números enteros (`int`) que indican la cantidad de elementos dentro del tablero. El primer entero corresponde cantidad de piezas explosivas, el segundo a la cantidad de peones, y el último entero corresponda a la cantidad de celdas vacías en el tablero.

  Ejemplo: si el tablero es uno de `(2, 3)` como el siguiente:
  `[["V2", "PP", "--"], ["H1", "--", "--"]]`, esta *property* retorna: `[2, 1, 3]`.

- **Modificar** `def peones_invalidos(self) -> int`:
  Property que utiliza el tablero guardado como atributo y cuenta la cantidad de peones que no respetan la Regla 3. Es decir, la cantidad de peones que tienen más de 1 vecino que es un peón.

  Ejemplo: si el tablero es uno de `(2, 3)` como el siguiente:
  `[["PP", "PP", "PP"], ["PP", "--", "--"]]`, esta *property* retorna `2` porque los peones de la posición `(0, 0)` y `(0, 1)` tienen más de 1 vecino.

- **Modificar** `def piezas_explosivas_invalidas(self) -> int`:
  Property que utiliza el tablero guardado como atributo y cuenta la cantidad de piezas explosivas que no son válidas para el tablero. Una pieza explosiva no es válida si la cantidad de celdas a destruir no equivale exactamente al alcance máximo que tenga dicha pieza. Para calcular el alcance máximo, se debe ignorar la existencia de los peones.

  Ejemplo: si el tablero es uno de `(2, 3)` como el siguiente:
  `[["H3", "V3", "PP"], ["R6", "PP", "R5"]]`, esta *property* retorna `2` porque las piezas `"V3"` y `"R6"` deben destruir más celdas de las que es posible en dicho tablero.

- Modificar `def tablero_transformado(self) -> list:`
  Property que retorna el mismo tablero que se tiene guardado como atributo, pero cada pieza explosiva es reemplazada por una instancia de la clase `PiezaExplosiva` con sus argumentos correspondientes.

  Ejemplo: si el tablero es uno de `(2, 2)` como el siguiente: `[["--", "V3"], ["--", "PP"]]`, esta *property* retorna: `[["--", PiezaExplosiva(3, "V", 0, 1)], ["--", "PP"]]`

- Modificar `def celdas_afectadas(self, fila: int, columna: int) -> int:`
  Método que utiliza el tablero guardado como atributo, y dada la posición de una pieza explosiva `(fila, columna)`, debe retornar la cantidad de celdas que son afectadas por la explosión de dicha pieza. No olvides que, según la Regla 1, una celda es alcanzada por la explosión si es que están en la misma fila, columna o diagonal de la pieza explosiva (esto depende del tipo de pieza), y que no existe un peón entre la pieza explosiva y la celda a destruir.

  En caso de que el tablero no contenga un pieza explosiva en la posición indicada, `(fila, columna)`, este método debe retornar `-1`.

- Modificar `def limpiar(self) -> None:`
  Método que actualiza el tablero de la instancia para eliminar todos los peones de dicho tablero. De este modo, el atributo quedará únicamente con celdas vacías y piezas explosivas.

- Modificar `def reemplazar(self, nombre_nuevo_tablero: str) -> bool:`
  Método que utiliza el archivo `tableros.txt` para extraer la información del tablero cuyo nombre sea igual al indicado en el argumento `nombre_nuevo_tablero`. Luego, actualiza los atributos de la instancia, `tablero` y `dimensiones`, con la información de este nuevo tablero. El tablero cargado debe ser almacenado con el mismo formato que el original, es decir, una lista de listas, donde cada celda es el *string* correspondiente. Luego de actualizar los atributos correctamente, este método debe retornar `True`.

  En caso que el tablero solicitado no exista en el archivo `tableros.txt`, este método debe retornar `False` y no sobre-escribir ningún atributo.

- Modificar `def solucionar(self) -> list:`
  Método que soluciona el tablero de la instancia. Para generar dicha solución, se deben posicionar los peones que sean necesarios para que cada pieza explosiva cumpla la Regla 1 y cada peón cumpla la Regla 2 y 3. Si el tablero ya incluye peones, estos no pueden ser retirados o desplazados a otra celda. La solución se debe generar a partir del tablero entregado. **Importante**: este método genera un nuevo tablero solucionado, no modifica el tablero original.

  La solución a entregar debe ser una lista de lista de *strings*. En caso que el tablero a solucionar no presente solución, debes retornar una lista vacía.

3.2. Parte 2 - Menú (30 %)

En esta parte deberás implementar un menú que permita interactuar con un tablero. Para esto, se ejecutará tu programa mediante un archivo `main.py`. Esta ejecución deberá aceptar argumentos de línea de comando para poder indicar el nombre del usuario que está usando tu programa y el nombre del tablero que se desea utilizar. Por ejemplo, en la consola, se escribirá "`python3 main.py StarlightAnya tablero_facil`". Lo que implica que el usuario se llama `"StarlightAnya"` y el tablero a utilizar será el de nombre `"tablero_facil"`. Para trabajar con argumentos de línea de comando, te recomendamos investigar sobre el funcionamiento de `sys.argv` de Python en internet.

Respecto a los argumentos de línea de comando, puedes asumir que siempre será una palabra por argumento, que el primero será el nombre del usuario y que el segundo será el nombre del tablero. No obstante,

debes realizar las siguiente validaciones:

- El nombre de usuario debe tener, al menos, 4 caracteres y solo debe contener letras del alfabeto. En caso que no se cumpla esta restricción, se debe imprimir, en consola, que el nombre es inválido.
- El nombre del tablero debe existir en la base de datos [1]. En caso que no se cumpla esta restricción, se debe indicar, en consola, que el tablero no existe.

La validaciones de estos 2 puntos, son independientes. Por ejemplo, si se entrega un nombre inválido y un tablero inexistente, se deben notificar ambos mensajes. En caso de levantar uno o los dos mensajes, el programa debe finalizar. En caso contrario, es decir, se cumplan ambas validaciones, se debe abrir el Menú de Acciones con sus opciones correspondientes.

3.2.1. Menú de Acciones

Este menú permitirá al usuario poder interactuar con el tablero elegido. Este menú debe saludar al usuario, utilizando el nombre de usuario indicado al momento de ejecutar el programa, y debe incluir las opciones de **mostrar tablero, limpiar tablero, solucionar el tablero y salir del programa**. Este menú debe ser a prueba de errores ante todo *input* ingresado por el usuario.

En esta parte de la tarea, se evaluará principalmente que (1) el menú sea aprueba de errores, (2) incluya todas las opciones solicitadas y (3) que cada opción del menú llame, mediante código, a la función o método correspondiente. No se evaluará si la opción seleccionada realmente hace lo esperado, eso será evaluado en la Parte 1.

A continuación, se detalla la función que debe cumplir cada una de las acciones que podrá realizar el usuario sobre el tablero escogido:

1. **Mostrar tablero**: Imprime el tablero en la consola. Se debe hacer uso de la función entregada en el archivo `imprimir_tablero.py`.
2. **Limpiar tablero**: Se encarga de eliminar todo peón del tablero para dejar únicamente las piezas explosivas. Una vez ejecutada esta opción, el tablero a utilizar en las siguientes opciones debe ser el generado por esta opción.
3. **Solucionar tablero**: Completa el tablero asegurando que se cumplan las Reglas 1, 2 y 3.
4. **Salir del programa**: Termina la ejecución del programa.

A continuación se entrega un ejemplo de cómo se podría ver este menú:

```
Hola StarlightAnya!

*** Menú de Acciones ***

[1] Mostrar tablero
[2] Limpiar tablero
[3] Solucionar tablero
[4] Salir del programa

Indique su opción (1, 2, 3 o 4):
```

Figura 4: Ejemplo de Menú de Acciones

[1] Más información en la sección de Archivos

4. Archivos

4.1. Código inicial

Para esta tarea, se te hará entrega de diversos archivos que deberás completar con funcionalidades. Puedes crear más archivos si lo estimas conveniente.

- Modificar `pieza_explosiva.py`: Aquí encontrarás la definición básica de la clase `PiezaExplosiva` que debes completar.
- Modificar `tablero.py`: Aquí encontrarás la definición básica de la clase `Tablero` que debes completar.
- Modificar `main.py`: Este será el archivo que será ejecutado para levantar el menú por consola.
- No modificar `tests/**.py`: Esta carpeta contendrá una serie de archivos `.py` que corresponden a diferentes *tests* para evaluar 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.
- No modificar `tests_todos.py`: Este archivo se encarga de ejecutar todos los *tests* contenidos en la carpeta `tests`.
- No modificar `imprimir_tablero.py`: Este archivo contiene la función `imprimir_tablero` que deberás ocupar en esta tarea. Más información en la sección de Archivos.
- No modificar `tableros.txt`: Este archivo contiene posibles tableros a cargar para la parte 2 de esta tarea.

4.2. `tableros.txt`

Para poder entender el formato de los tableros, se te facilitará un archivo que contendrá información sobre los diferentes tableros que se pueden utilizar.

Dentro del archivo `tableros.txt` podrás encontrar información de los posibles tableros del *DCChexxploding*. El contenido de este archivo estará dado por diversas líneas de texto, donde cada línea tendrá el siguiente formato:

- El primer elemento antes de la primera coma (`","`) representa el nombre (`string`) del tablero.
- Los siguientes dos elementos corresponden a dos números enteros mayores a 0, `N` y `M`, los cuales representan la dimensión tablero. El primer número (`N`) indica la cantidad de filas, mientras que el segundo número (`M`) indica la cantidad de columnas.
- Luego vendrán `NxM` elementos separados por una coma (`","`), donde por cada `N` elementos se interpretará como una fila del tablero. Cada uno de estos elementos representa una celda del tablero, el cual puede ser `"--"` que representa una celda vacía, un `"PP"` que representa un peón, o bien una pieza explosiva, la cual siempre partirá con el *string* `"V"`, `"H"` o `"R"` seguido por un número entero superior a 0.

A continuación se presenta un ejemplo de una línea del archivo y su representación en el tablero:

```
ejemplo_chico_sin_solución,2,5,V2,--,PP,--,H2,H3,--,--,PP,R11
```

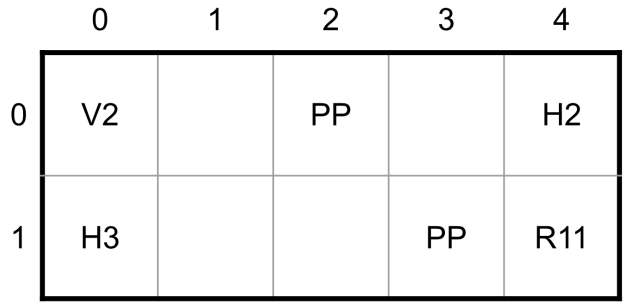

Figura 5: Representación del tablero `"ejemplo_chico_sin_solución"` en forma de tablero

4.3. `tablero.py`

Para facilitar tu trabajo, ~~los esclavos de **Gatochico**~~ **maestros ayudantes** te harán entrega del módulo `imprimir_tablero.py` que contiene la función `imprimir_tablero(tablero: list)` encargado de imprimir el tablero en pantalla de una forma ordenada. El parámetro `tablero` corresponde a una lista de listas con la información del tablero.

Será tu deber importar **correctamente** la función `imprimir_tablero(tablero: list)` y hacer uso de ella para imprimir el tablero en consola. A continuación se muestra un ejemplo del tablero y su representación en consola al llamar la función:

```
tablero = [
    ['V2', '--', 'PP', '--', 'H2'],
    ['H3', '--', '--', 'pp', 'R11']
]
imprimir_tablero(tablero)
```

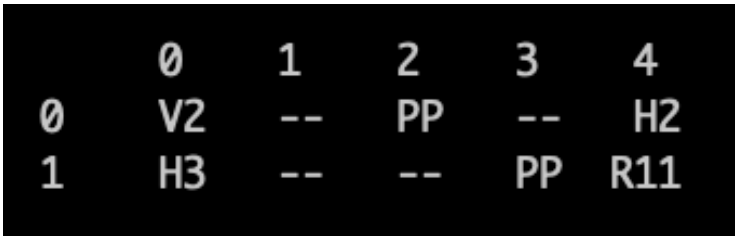

Figura 6: Tablero en consola

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 a ignorar para esta tarea son:

- El enunciado.
- Los archivos que no debes modificar: `imprimir_tablero.py`.
- La base de datos: `tableros.txt`.

- La carpeta `tests/`.

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 esta señalará con color **amarillo** 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. 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.

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. Luego, todo ítem corregido automáticamente será evaluado de forma binaria: pasa todos los *tests* de dicho ítem o no los pasa todos. Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente.

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**.

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 3.10.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- 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.
- 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).