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

IIC2233 Programación Avanzada (2024-2)

Tarea 1

Entrega

- **Fecha y hora oficial (sin atraso):** lunes 26 de agosto de 2024, 20:00.
- **Fecha y hora máxima (2 días de atraso):** miercoles 28 de agosto de 2024, 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.
- 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. *DCCultivo*

El reconocido y popular profesor **Hernán** ha decidido colgar el teclado y el *mouse* por unos meses. Cansado de la ciudad, los colapsados medios de transporte, y sobre todo los retos de **Cruz**, busca contemplar una vida más tranquila y pausada. Es por eso que decide tomar todos sus ahorros e invertirlos en un campo de cultivo donde podrá disfrutar de la vida bucólica[1].

Sin embargo, tras varios meses de trabajo se da cuenta que no es una tarea fácil. Lo peor de todo, las horas utilizadas en planificar su cultivo le impiden realizar su actividad favorita: ¡ver anime!

Es por eso que una mañana decide publicar en sus redes sociales una interesante propuesta 100 % de su autoría: *DCCultivo*. Quien acepte este proyecto ayudará a **Hernán** a modelar la disposición y riego de varias especies de cultivos en su predio usando el programa *DCCultivo*. Poniendo en práctica todo lo aprendido en "IIC1103 - Introducción a la programación" decides ayudar a **Hernán** en su nueva etapa de vida para que él pueda dar largas caminatas, disfrutar de mucho anime y apreciar su nueva vida en el campo al máximo.

2. Predios de cultivo

Para poder desarrollar el sistema *DCCultivo*, deberás simular la ocupación de terreno en varios predios de cultivo de dimensiones limitadas. Para esto, ocuparemos una forma de trabajar el terreno de cada predio como una matriz. Esta matriz puede tener forma rectangular, con `n` unidades de alto y `m` unidades de ancho. También puede ser cuadrada, con igual cantidad de unidades de alto y de ancho. Cada celda dentro de la matriz representa un espacio que puede tener un número o una letra `"X"`, dependiendo si existe o no un cultivo en ese espacio.

El alto de una matriz será la cantidad total de filas, y el ancho de esta la cantidad total de columnas. Para referirnos a una fila o columna de la matriz, utilizaremos la notación ($fila_i$, $columna_j$) donde i y j son números enteros mayores o igual a 0 que indican la fila o columna de la matriz. Por ejemplo, $fila_0$ hace referencia a la primera fila de la matriz, la fila número 0; y $columna_2$ hace referencia a la tercera columna de la matriz, la columna número 2. Por temas de estandarización, para todas las matrices el origen de la matriz estará ubicado 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 (alto - 1, ancho - 1).

A continuación en la Figura 1 se muestra un único predio de 5 de alto por 7 de ancho, es decir cuenta con un total de 5 filas y 7 columnas. A la izquierda se muestra el predio con todas sus celdas vacías, las cuales se indican con el *str* `"X"` siempre en mayúscula. Como se muestra a la derecha, los predios pueden ser plantados con cultivos representados por códigos que van desde el *int* `0` al `9`. Es decir, sólo existen 10 tipos de cultivos en total. La forma de plantar estos cultivos es por bloques, lo cual será explicado en la sección Plantado de cultivos.

	0	1	2	3	4	5	6
0	X	X	X	X	X	X	X
1	X	X	X	X	X	X	X
2	X	X	X	X	X	X	X
3	X	X	X	X	X	X	X
4	X	X	X	X	X	X	X

Predio P1 vacío

	0	1	2	3	4	5	6
0	3	3	3	1	1	9	9
1	3	3	3	1	1	9	9
2	3	3	3	X	X	9	9
3	7	7	7	7	7	9	9
4	7	7	7	7	7	9	9

Predio P1 con cultivos

Figura 1: Matriz que representa un predio de 5 de alto por 7 de ancho, primero vacío y luego con los cultivos de códigos 1, 3, 7, y 9.

[1] adj. Que evoca de modo idealizado el campo o la vida en el campo.

El campo adquirido por **Hernán** se divide en varios predios de diferentes dimensiones, los cuales se distinguen entre ellos gracias a un código único representado por la letra P seguido de un número (ej: P1, P3, P15). Al comenzar el programa y a través de una interacción por menú, se indicará la opción de generar los predios los cuales iniciarán vacíos. Para efectos de esta evaluación, **Hernán** entregó un archivo donde tiene indicados los códigos únicos de los predios y sus tamaños correspondientes. Más información de estos archivos se encuentran en la sección de Archivos.

2.1. Plantado de cultivos

Una de las funcionalidades principales de *DCCultivo* es mantener tanto el orden como el inventario de cultivos. Para ello, se define un sistema de plantado por bloques en donde se priorizará siempre insertar el bloque de cultivo partiendo en **la primera celda superior izquierda** disponible dentro del predio. Al intentar plantar cultivos en un predio se deben recibir primero un número *int* relativo al código de cultivo (`0` al `9`), luego el alto y finalmente el ancho del bloque. No se permitirá rotación del bloque en ningún eje, debiendo respetar las proporciones originales.

Como se muestra en la Figura 2, la acción de plantar se puede realizar de forma sucesiva. En este ejemplo se plantan 4 cultivos distintos, los cuales modifican la matriz al ser plantados uno tras otros. Se busca y luego insertan los números respectivos al código del cultivo en la primera **fila** disponible, siempre lo más a la izquierda que se pueda, y que pueda abarcar el bloque completo.

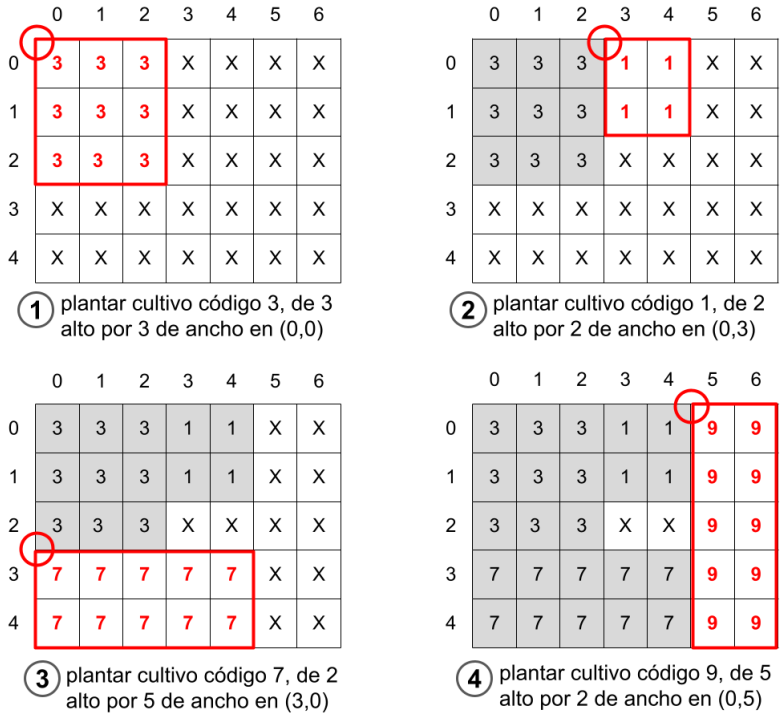

Figura 2: Ejemplo de secuencia de plantado de cultivos según espacio disponible en el predio P1.

Recordemos que se dispone de diferentes predios ordenados por código. Una restricción importante es que no se permite repetición del código de cultivo en el mismo predio. Además, en el caso de que el bloque de cultivo pedido no pueda ser plantado en el primer predio ó que ya se encuentre el código de cultivo en ese predio, se debe buscar otro predio que sí pueda cumplir con lo pedido. Dado esto, el bloque se intentará ubicar en el segundo predio, en el tercero, y así sucesivamente hasta encontrar espacio como se muestra en la Figura 3. Pueden existir casos donde no se pueda colocar el bloque pedido en ningún predio. Al

buscar dónde plantar los cultivos, se debe respetar el orden de prioridad de los predios disponibles, desde el primero hasta el último generado según el archivo.

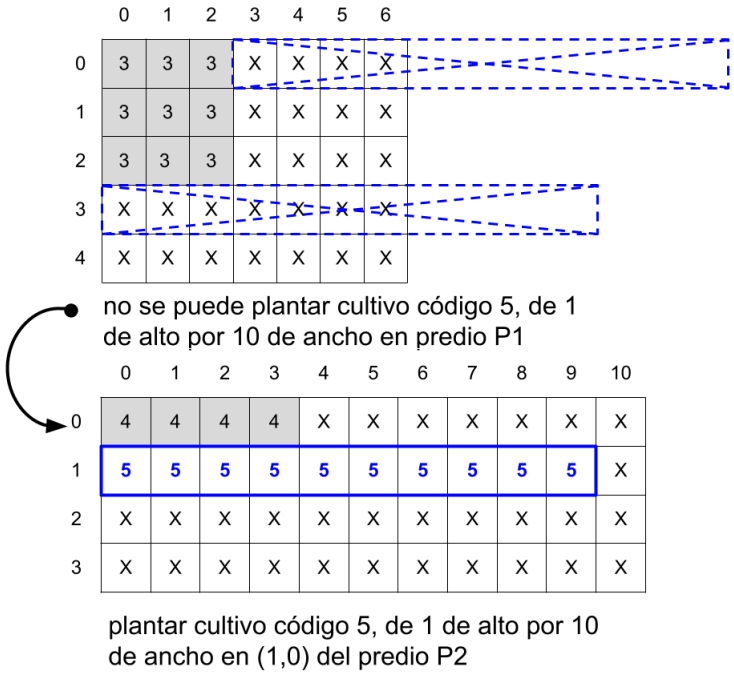

Figura 3: Ejemplo de búsqueda.

2.2. Planos de riego

Como bien dice el profesor **Hernán**, tomar agua es una de las acciones más importantes de tu día como estudiante. Ahora traspasará sus consejos a los cultivos de su campo.

En *DCCultivo*, cada predio contempla un plano distinto y especializado capaz de visualizar cómo ha sido realizado su riego en el tiempo. Este es el plano de riego. Todos los planos de riego inician con el valor *int* `0` en todas sus celdas, es decir inician sin haber sido regados. Al momento de regar el predio, se aumenta en 1 unidad de agua toda un área circular respecto a un punto central. El sistema debe recibir las coordenadas $(fila_i, columna_j)$ del centro del círculo y el *int* correspondiente a la extensión de su área. Un área igual a 2 significa extender el círculo 2 unidades más desde el centro. Para simplificar, se considerará el dibujo de un círculo dentro de la matriz como si fuera un cuadrado que contenga un círculo inscrito, excluyendo las 4 celdas correspondientes a sus vértices. La Figura 4 demuestra cómo se visualiza el área de riego para este programa, en donde las esquinas excluidas se demarcan en gris.

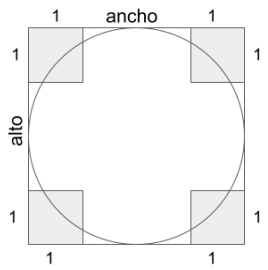

Figura 4: Un cuadrado que no considera sus esquinas.

La acción de regar siempre incrementará en 1 unidad la celda correspondiente, por lo que en el caso de que una de las celdas dentro del círculo ya se encuentre regada, se aumentará en 1 su valor, como se ve en la Figura 5. Es importante notar que en el caso de que el círculo de riego sobrepase el predio, sólo se deberá aumentar en 1 las celdas dentro los límites alcanzables de dicho predio dentro del círculo. Es decir, no se extenderá más allá de este. Además, se puede asumir que las coordenadas del centro del círculo siempre estarán dentro del predio.

Figura 5: Ejemplo de plano de riego tras dos acciones de riego sucesivas.

2.3. Control de plagas

A pesar de todos los esfuerzos de **Hernán** por aislarse de su vida de profesor, **Cruz** encuentra la forma de arruinar su día a día[2]. **Cruz** ha descubierto el punto sensible de *DCCultivo*: el control de plagas. Actualmente no existe ningún sistema de defensa que impida la proliferación de estas insidiosas y molestas plagas. Es por esto que junto a **Hernán** deciden diseñar un sistema de detección temprana, con el fin de eliminar los posibles cultivos afectados antes de que se expandan. Para ello contarás con una función especial que te entregará todas las coordenadas de todos los predios afectados. Si al menos una celda del cultivo se encuentra afectado por una de las plagas de **Cruz**, el bloque correspondiente a ese código debe removerse por completo del predio, reemplazando todos los números del código dentro del bloque por el *str* `"X"`.

[2]Secretamente Cruz realiza todas estas fechorías para convencer a Hernán de entrar a un doctorado.

3. Flujo del programa

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

Parte 1 Completar un archivo con las clases `Predio` y `DCCultivo`, las cuáles permitirán: abrir archivos, analizar, aplicar cambios y responder a diferentes consultas del programa.
Esta parte será corregida automáticamente mediante el uso de *tests*.

Parte 2 Confeccionar un menú por consola en dos partes, que permita interactuar mediante *DCCultivo* todos los predios del terreno y sus funcionalidades.
Esta parte será corregida manualmente por el cuerpo docente.

3.1. Parte 1 - Funcionalidades

En esta parte, deberás completar diversos métodos de las dos clases indicadas para trabajar con el sistema de *DCCultivo*. La corrección completa de esta parte será mediante *tests*. Es por esto que debes asegurarte que cada método se pueda ejecutar correctamente, que el archivo no presente errores de sintaxis, y no cambiar el nombre y ubicación del archivo.

Para cada clase y método indicado a continuación, **no puedes modificar su nombre, agregar nuevos argumentos o argumentos por defecto**. En caso de no respetar lo indicado, la evaluación presentará un fuerte descuento. 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. El requisito primordial es que debes mantener el formato de los métodos y atributos informados en este enunciado.

Adicionalmente, para apoyar el desarrollo de esta parte, se provee de una batería de *tests* donde podrán revisar distintos casos con su respuesta esperada. Estos *tests* públicos corresponden a un segundo chequeo de la evaluación, es decir, no son representativos de todos los casos posibles que tiene una función/método. 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 van en contra de lo expuesto en el enunciado.**

A continuación se describen clases `Predio` y `DCCultivo` que deberás completar. **Importante**: para todos los métodos de clase donde se deba abrir un archivo, puedes asumir que el formato será adecuado y que tanto la cantidad como los tamaños de los predios **podrían cambiar en el momento de la evaluación**.

Modificar `class Predio:`
Clase que representa un único predio.

- No modificar `def __init__(self, codigo_predio: str, alto: int, ancho: int) -> None:`
  Inicializa una instancia Predio y asigna los siguientes atributos:

`self.codigo_predio`	Un `str` que representa el código único para ese predio.
`self.alto`	Un `int` que representa cuántas filas tiene el predio.
`self.ancho`	Un `int` que representa cuántas columnas tiene el predio.
`self.plano`	Una lista de listas que contiene todas las filas y columnas del predio. Inicia como una lista simple vacía `[]`.
`self.plano_riego`	Una lista de listas que contiene todas las filas y columnas del predio. Inicia como una lista simple vacía `[]`.

- Modificar `def crear_plano(self, tipo: str) -> None:`
  La primer acción básica es poder generar las matrices que representarán los predios de cultivo. Todas las matrices se trabajarán como listas de listas, donde la cantidad de listas es la cantidad de filas totales y cada elemento de cada lista es una columna. Como existen dos tipos de planos, el método deberá recibir un *str* que indique el tipo de plano a crear, siendo `"normal"` ó `"riego"`. Para cada tipo de plano se deben tener las siguientes consideraciones al crearlo:
  - `"normal"`: Se debe modificar el atributo original `self.plano`, para que cada elemento de cada lista de listas sea el *str* `"X"`.
  - `"riego"`: Se debe modificar el atributo original `self.plano_riego`, para que cada elemento de cada lista de listas sea el *int* `0`.
- Modificar `def plantar(self, codigo_cultivo: int, coordenadas: list, alto: int, ancho: int) -> None:`
  Este método se encarga sólo de plantar el bloque de cultivo en el predio según lo indicado en Plantado de cultivos. Al ser llamado, este modificará el atributo `self.plano` reemplazando las celdas vacías (`"X"`) con el número respectivo al código de cultivo.

  Recordemos que los cultivos deben agregarse por bloques, por lo que el método recibirá además, las coordenadas de posicionamiento del bloque como lista de *int* $[fila_i, columna_j]$, donde el primer elemento de la lista es el número de fila y el segundo el número de columna. Seguido a eso recibirá el alto y el ancho correspondiente a la cantidad de filas y columnas respectivas que abarcará el bloque.

  Puedes asumir que siempre se entregarán cordenadas de posicionamiento válidas, y que el alto y ancho del bloque siempre va a caber dentro del predio.
- Modificar `def regar(self, coordenadas: list, area: int) -> None:`
  Este método se encarga de regar los cultivos según lo indicado en Planos de riego. Al ser llamado, este modificará el atributo `self.plano_riego` aumentando los valores de las celdas en 1 según la zona regada. Las coordenadas del centro del círculo se reciben como lista de *int* $[fila_i, columna_j]$, donde el primer elemento de la lista es el número de fila y el segundo el número de columna. Puedes asumir que las coordenadas entregadas siempre estarán dentro del predio, y que el área a regar será siempre mayor o igual a 1.
- Modificar `def eliminar_cultivo(self, codigo_cultivo: int): -> int:`
  Este método se encargará de eliminar el código de cultivo del predio, reemplazando los valores del bloque por el *str* `"X"`. Finalmente debe retornar la cantidad de celdas eliminadas que conformaban el bloque como *int*. Puedes asumir que siempre se entregarán códigos de cultivo que ya estén dentro del predio.

Modificar `class DCCultivo:`
Clase que representa el sistema de organización de predios, manejo de los cultivos y detección de plagas.

- No modificar `def __init__(self) -> None:`
  Inicializa una instancia de DCCultivo y guarda como atributo una lista vacía correspondiente a los predios que serán completados a futuro.
  - `self.predios`: Lista inicialmente vacía, se encargará de almacenar objetos de la clase `Predio`.
- Modificar `def crear_predios(self, nombre_archivo: str) -> str:`
  Método que carga el archivo de texto `nombre_archivo` para extraer la información de todos los predios. Este argumento ya incluye la extensión del archivo. Debe actualizar el atributo `self.predios`, agregando instancias de la clase `Predio` con sus planos ya creados. El orden de los objetos en la lista seguirá el orden del archivo.

Finalmente este método debe retornar el mensaje `"Predios de DCCultivo cargados exitosamente"`. En caso que el archivo solicitado no exista, este método debe retornar el mensaje: `"Fallo en la carga de DCCultivo"` y no modificar ningún atributo.

- **Modificar** `def buscar_y_plantar(self, codigo_cultivo: int, alto: int, ancho: int) -> bool`

  Esta es una de las partes fundamentales de *DCCultivo*. Según lo indicado en Plantado de cultivos, deberá recibir las características del bloque de cultivo a plantar, realizar la búsqueda del predio idóneo y plantar el bloque en éste. Como consecuencia se modificará el atributo `plano` de uno de los predios del atributo `self.predios`. Finalmente deberá retornar el *bool* `True` si es que pudo ubicarse el bloque pedido en uno de los predios. En caso de que ningún predio cumpla con las condiciones, retornará `False`.

- **Modificar** `def buscar_y_regar(self, codigo_predio: str, coordenadas: list, area: int) -> None:`

  Según lo indicado en Planos de riego, este método deberá buscar y regar el predio solicitado. Como consecuencia se modificará el atributo `self.plano_riego` de este predio contenido en el atributo `self.predios`.

- **Modificar** `def detectar_plagas(self, lista_plagas: list[list]) -> list[list]:`

  Método encargado de modificar uno o varios de los predios del atributo `self.predios`. Recibirás una lista de listas con los códigos de los predios y las coordenadas donde existen plagas de la siguiente forma: `[[codigo_predio, [fila, columna]], ...]`. Dentro de la lista de listas pueden repetirse códigos de predio, pero no se repetirán las mismas coordenadas dentro del mismo predio. Como se explica en Control de plagas el método debe realizar una búsqueda y eliminar los bloques de códigos de cultivos de los predios donde al menos una (1) celda de ese bloque de cultivo se encuentre afectada por la plaga.

  Finalmente, el método debe retornar una lista de listas con información de todos los predios afectados por la plaga y que sufrieron eliminación de bloques. Cada una de las sub listas contiene dos elementos: el primero corresponde al código del predio, mientras que el segundo corresponde a la cantidad de celdas eliminadas en dicho predio. El orden de la lista de listas a retornar debe ser de forma ascendente según la cantidad de celdas eliminadas y, en caso de que haya empate entre dos o más, estos se deberán ordenar alfabéticamente según el código del predio.

  Ejemplo: Si tras la plaga al predio P1 se le eliminaron 100 celdas de cultivos en total, en el predio P3 se eliminaron 30 celdas de cultivos, y en el predio P4 se eliminaron 100 celdas de cultivos, la lista de listas a retornar se verá así: `[["P3",30],["P1",100],["P4",100]]`

3.2. Parte 2 - Menú

En esta parte deberás implementar un menú que permita interactuar con los predios del campo y varias de las funcionalidades de *DCCultivo*. Para esto, se ejecutará tu programa mediante un archivo `main.py`. Las diversas acciones a ejecutar en los menús deberán aceptar entradas de usuario mediante terminal.

La interacción del menú se divide en dos partes: Menú de Inicio 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 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, 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.

3.2.1. Menú de Inicio

Este es el menú encargado de iniciar la interacción con el usuario y realizar la carga de *DCCultivo*. 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 DCCultivo!

*** Menú de Inicio ***

[1] Crear predios
[2] Salir del programa

Indique su opción (1, 2):
```

Figura 6: Ejemplo de Menú de Inicio

A continuación, se detalla la función que debe cumplir cada una de las acciones que podrá realizar el usuario en el Menu de Inicio:

1. **Crear predios**: Se encarga de usar el método `crear_predios` de la clase `DCCultivo`. Luego de esto, se debe continuar inmediatamente al Menú de Acciones.
2. **Salir del programa**: Termina la ejecución del programa.

3.2.2. Menú de Acciones

Luego de haber seleccionado la opción 1 del Menú de Inicio, se abrirá un segundo menú encargado de realizar las acciones sobre los predios cargados. A continuación se entrega un ejemplo de cómo se podría ver el Menú de Acciones:

```
¡Bienvenido a DCCultivo!
Tu terreno está listo para trabajar.

*** Menú de Acciones ***

[1] Visualizar predio
[2] Plantar
[3] Regar
[4] Buscar y eliminar plagas
[5] Salir del programa

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

Figura 7: Ejemplo de Menú de Acciones

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

1. **Visualizar predio**: Imprime los planos de un predio específico en la terminal. Para ello deberás utilizar la función `imprimir_planos` que se te proveerá en el archivo `utils.pyc`, la cual recibe como parámetro un objeto de la clase `Predio`.

   Deberás solicitar al usuario el *str* correspondiente al código del predio a visualizar, utilizando la función `input()`. Luego, con esa información podrás buscar objeto `Predio` correspondiente dentro de `DCCultivo`. En caso de ingresar un código no existente, se debe notificar por terminal. Finalmente se debe regresar al Menú de Acciones.

2. **Plantar**: Se encarga de utilizar el método `buscar_y_plantar` de la clase `DCCultivo`. Para los parámetros `codigo_cultivo`, `alto` y `ancho` deberás utilizar la función `input()` para solicitar al usuario que ingrese los tres datos. Puedes asumir que el usuario ingresará correctamente los datos, es decir, entregará un *str* para el parámetro de `codigo_cultivo` y dos números enteros positivos para los parámetros de `alto` y `ancho`. El orden y formato para pedir los tres datos quedan a criterio del programador mientras se utilice la función `input()`. Finalmente, se deberá imprimir en consola un mensaje de éxito o fallo dependiendo del resultado retornado por dicho método, y regresar al Menú de Acciones.

3. **Regar**: Se encarga de utilizar el método `buscar_y_regar` de la clase `DCCultivo`. Para los parámetros `codigo_predio`, `coordenadas` y `area` deberás utilizar la función `input()` para solicitar al usuario que ingrese los tres datos. Puedes asumir que el usuario ingresará correctamente los datos. El orden y formato para pedir los datos, incluida la lista de coordenadas, queda a criterio del programador mientras se utilice la función `input()`. Finalmente debe regresar al Menu de Acciones.

4. **Buscar y eliminar plagas**: Gatilla el sistema de búsqueda y detección de plagas. Para ello, deberás primero extraer la información de las plagas usando la función `plagas` que se te proveerá en el archivo `utils.pyc`. La lista retornada se entregará como parámetro al método `detectar_plagas` de la clase `DCCultivo`. Finalmente, se deberá imprimir en consola el resultado retornado por dicho método, junto a un mensaje de éxito, y regresar al Menú de Acciones.

5. **Salir del programa**: Termina la ejecución del programa.

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 `dccultivo.py`: Aquí encontrarás la definición básica de las clases `Predio` y `DCCuidad` que debes completar.
- Crear `main.py`: Este será el archivo que será ejecutado para levantar el menú por consola.
- No modificar `utils.pyc`: Este archivo contiene las funciones `imprimir_planos` y `plagas` que deberás ocupar en esta tarea. Más información de este archivo en la sección de `utils.pyc`.
- No modificar `data/`: Esta carpeta contendrá un archivo `predios.txt` que corresponde a los códigos de predios y sus dimensiones. El contenido de este archivo podrá cambiar al momento de evaluar, manteniendo el formato.
- 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. `data/predios.txt`

Para poder entender el formato de los diferentes predios en el campo, se te facilitará de un archivo que contendrá información sobre todos los predios disponibles. Recordar que el contenido de este archivo podrá cambiar al momento de evaluar, manteniendo el formato.

Dentro de la carpeta "`data/`" encontrarás el archivo `predios.txt`. El contenido este archivo estará dado por diversas líneas de texto, en donde cada línea contiene la información de cada predio separada entre comas. Todas las líneas y sus elementos tendrán siempre el mismo formato:

- El primer elemento será un *string* que representa el código de cada predio.
- El segundo elemento será un *string* que representa el alto del predio como número entero positivo.
- El tercer elemento será un *string* que representa el ancho del predio como número entero positivo.

A continuación se presenta un ejemplo del archivo "`predios.txt`" y su representación visual como lista de listas para ambos planos de cultivo y de riego, resultante de llamar a la función `imprimir_planos` de `utils.pyc`:

```
P1,5,6
P2,6,9
P3,3,8
```

```
**** Predio P1 ****
0 celdas cultivadas
0 unidades de riego

-- plano de cultivos --
[["X","X","X","X","X","X"],
["X","X","X","X","X","X"],
["X","X","X","X","X","X"],
["X","X","X","X","X","X"],
["X","X","X","X","X","X"]]
-- plano de riego --
[[0,0,0,0,0,0],
[0,0,0,0,0,0],
[0,0,0,0,0,0],
[0,0,0,0,0,0],
[0,0,0,0,0,0]]
```

```
**** Predio P2 ****
0 celdas cultivadas
0 unidades de riego

-- plano de cultivos --
[["X","X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X","X"]]
-- plano de riego --
[[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0]]
```

```
**** Predio P3 ****
0 celdas cultivadas
0 unidades de riego

-- plano de cultivos --
[["X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X"],
["X","X","X","X","X","X","X","X"]]
-- plano de riego --
[[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0]]
```

4.3. `utils.pyc`

Para facilitar tu trabajo y evaluar el uso de módulos, se te hará entrega del módulo `utils.pyc` que contiene dos funciones que deberás utilizar en el desarrollo de 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: 3.11.X con X mayor a 7. Utilizar una versión distinta a 3.11 provocará un error de "magic number". Solo es necesario realizar `import utils` desde otro archivo `.py` para utilizar sus funciones.

Las funciones que contiene `utils.pyc` son:

- `imprimir_planos(predio: Predio) -> None:`
  Esta función recibe una instancia de la clase Predio. Luego, se encargará de imprimir en consola información relevante del predio junto a sus planos.
- `plagas(dccultivo: DCCultivo) -> list[list]:`
  Esta función recibe una instancia de la clase DCCultivo. Retorna una lista de listas con los códigos de los predios y las coordenadas donde existen plagas de la siguiente forma:
  `[[codigo_predio, [fila, columna]], ...]`. Pueden existir repeticiones de predios, pero no se repetirán las coordenadas dentro de los mismos predios.

Será tu deber importar **correctamente** este archivo y hacer uso de sus funciones para el desarrollo de la tarea. Recuerda que **no** debes modificar su contenido.

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 `utils.pyc`
- 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, corroborando que cada archivo de *tests* individual que les pasamos **corra en un tiempo acotado de 10 segundos**, en caso contrario se asumirá un resultado incorrecto.

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>.py`
  Reemplazando `<test_N>` por el test que desees probar.

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

- `python3 -m unittest -v -b tests_publicos.test_00_crear_plano.py`

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `python`, `py3` o `python3.11`.

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 3.11.X con X mayor o igual a 7.
- 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).