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

IIC1103 – Introducción a la Programación
2 - 2016

Enunciado Tarea 3

Recordatorio:

- **Fecha de entrega: 17 de noviembre del 2016 a las 23:59 hrs.**
- **Foro de consulta:** `https://goo.gl/jw2x43`
- **Este trabajo es estrictamente personal. Recuerda leer la Política de Integridad Académica del DCC disponible en** `http://www.ing.uc.cl/ciencia-de-la-computacion/programas/licenciatura/politica-de-integridad-academica/`**. Se usará un software anti-plagio para detectar la copia (similitud entre códigos).**
- **Esta es una instancia formal de evaluación. Tu respuesta será leída por otros, por lo que debes utilizar un lenguaje apropiado. Además, no se aceptarán comentarios que no estén relacionados a la tarea.**
- **Recuerda subir a Classter avances parciales, puedes subir cuantas veces quieras el archivo de tu tarea. Se evaluará la última versión subida.**

¡Atención!

Ten en consideración que **no se recibirán entregas fuera del plazo**. Tampoco se responderá si te equivocas en entregar el archivo.
Es de tu responsabilidad ir subiendo entregas parciales de tu tarea. Se revisará la última versión que hayas subido al sitio web del curso.

Objetivo

En esta tarea, se espera que practiques los contenidos de control de flujo, ciclos, strings, listas, y especialmente archivos, Programación Orientada a Objetos (POO) y recursión. Para esto, deberás realizar un programa que permita a un usuario completar un puzzle llamado Akari.

El juego: Akari

Akari es un puzzle también conocido como *light up* o iluminar, pues consiste en poner ampolletas sobre un tablero tal que se "iluminen" todas las celdas de éste. Puedes jugar online en `http://www.nikoli.com/en/puzzles/bijutsukan/index.html`. A continuación, se explican las reglas del juego.

Objetivo del juego

Este juego consiste en un tablero de dimensión variable no necesariamente cuadrado donde se deben colocar ampolletas que iluminen dicho tablero. Existen dos tipos de celdas: negras y blancas. Las celdas negras pueden contener información sobre la cantidad de ampolletas que se pueden poner en celdas adyacentes que compartan un borde con ésta. Además, estas celdas "bloquean"el paso de la luz. Las celdas blancas son aquellas donde el usuario puede poner ampolletas, las que iluminan todas las celdas que se encuentran en la misma columna y fila y que no están separadas por una celda negra. Al colocar correctamente todas las ampolletas, todas las celdas blancas estarán "iluminadas".
En la Figura 1 se muestra un tablero inicial. Puedes observar que solo algunas celdas negras contienen información sobre la cantidad de ampolletas que pueden estar en alguna de las 4 celdas adyacentes que comparten un borde con la celda.
Nota: Puedes asumir que todos los tableros que consideraremos para esta tarea tienen una única solución

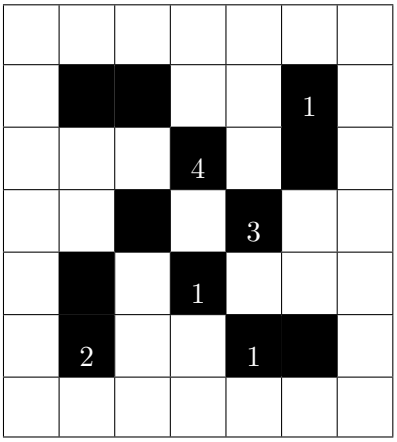


Figura 1: Tablero inicial

En el ejemplo mostrado en la Figura 1, las celdas marcadas en negro son celdas negras, las celdas que tienen números son celdas con restricción y las celdas blancas son las celdas donde el usuario puede colocar ampolletas.

Cada vez que el usuario quiera poner una ampolleta en una celda blanca, debe tener en consideración lo siguiente:

1. Las ampolletas solo se pueden poner en celdas blancas, las que pueden estar adyacentes a celdas negras o no. El número que aparece en las celdas negras dicen **exactamente** el total de ampolletas que se encuentran adyacentes (es decir, que comparten un borde) a ella, de manera vertical y horizontal.

2. Cada ampolleta ilumina la celda en la que está y además en dirección horizontal y vertical desde su celda hasta una celda negra o el borde del tablero .

3. Una ampolleta no puede iluminar a otra ampolleta, es decir, no puede haber más de una ampolleta en la misma fila o columna.

En la Figura 2 se muestra la solución del ejemplo mostrado en la Figura 1, las ampolletas están marcadas por el símbolo ☼. Adicionalmente y a modo de ejemplo, están pintadas con amarillo las celdas que se iluminan con la ampolleta ubicada en la celda (6,4):

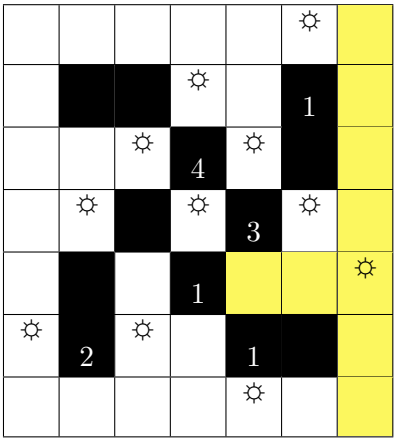


Figura 2: Solución al tablero inicial

Una estrategia para completar un tablero Akari es comenzar con las celdas que tienen una celda vecina con número y, una vez teniendo todas las celdas que tienen números con sus respectivas ampolletas puestas, continuar verificando el resto del tablero por si es necesario poner más ampolletas.

Tu programa

Tu programa debe estar implementado utilizando Programación Orientada a Objetos. Para esto, deberás definir la clase `Tablero` que representa un tablero de Akari. Además deberás programar, como mínimo, los métodos que se enuncian en la siguiente sección:

Métodos

Desde el método 2 en adelante, se considerara como ejemplo el puzzle Akari mostrado en la siguiente Figura, en donde las filas se cuentan de arriba hacia abajo y las columnas de izquierda a derecha, ambos partiendo desde el número 0:

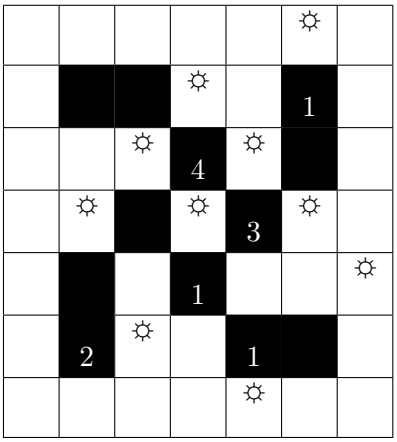


Figura 3: Tablero de ejemplo

Los métodos que deberá tener la clase `Tablero` son:

1. `__init__(self,nombre)`: lee el archivo `nombre` y deja almacenada la información de dicho tablero. El formato del archivo se muestra en la sección más abajo.

2. `tablero_completo(self)`: retorna `True` si el tablero está completo y `False` en caso contrario. Se entiende que el tablero está completo cuando todas las celdas están iluminadas. Considerando el tablero de ejemplo, este método debería retornar `False` porque aún hay celdas que no están iluminadas, como por ejemplo la celda (6,0).

3. `esta_iluminada(self,i,j)`: retorna `True` si la celda de la i-ésima fila y j-ésima columna se encuentra iluminada. Este método retorna `False` en caso contrario. Por ejemplo:

   - `esta_iluminada(2,0)` retorna `True` pues hay una ampolleta en la misma fila.
   - `esta_iluminada(4,5)` retorna `True` pues hay dos ampolletas que la iluminan: la que se encuentra en la cuarta fila y la que se encuentra en la quinta columna.
   - `esta_iluminada(6,0)` retorna `False`, pues no hay ninguna ampolleta en la sexta fila ni en la quinta columna.

4. `asignacion_valida(self,i,j)`: retorna `True` si se puede poner una ampolleta en la celda de la i-ésima fila y j-ésima columna considerando el estado actual del tablero. Por ejemplo:

   - `asignacion_valida(1,0)` retorna `True` pues no se viola ninguna restricción hasta el momento.
   - `asignacion_valida(4,4)` retorna `False` pues la celda negra (3,4) ya tiene tres ampolletas en celdas adyacentes.
   - `asignacion_valida(6,6)` retorna `False`, pues ya se encuentra una ampolleta en la misma fila y columna.
   - `asignacion_valida(5,0)` retorna `True`, pues no se viola ninguna restricción hasta el momento.

5. `resolver_tablero(self,...)`: resuelve de manera **recursiva** el tablero. Los atributos que tiene este método dependerá de la implementación que tenga, por lo que queda abierto a que tú decidas cuáles son estos atributos.

6. `tablero_resuelto(self)`: retorna `True` si el tablero está completo correctamente, `False` en caso contrario. Ten en cuenta que un tablero puede estar completo (retornar `True` en el método `tablero_completo`) pero no estar correctamente resuelto.

Si estimas conveniente, puedes implementar todos los métodos adicionales que sean de ayuda para resolver tu tarea. También puedes agregar todas las funciones que desees.

Archivos

A continuación se describe el formato de los archivos que debe leer tu programa. Para esto, se considerará el tablero de Akari mostrado en la Figura 3:

En el archivo, las celdas negras son representadas por una equis (`'X'`), las celdas negras con información solo tienen un número con la cantidad de ampolletas que debe tener alrededor. Las celdas blancas son representadas con un guión (`'-'`). Por último, las celdas blancas que tienen una ampolleta están representadas por un asterisco (`'*'`)
Todas las filas están en una línea. Las columnas están separadas por una coma (`','`). Considerando el ejemplo, el archivo correspondiente es el siguiente:

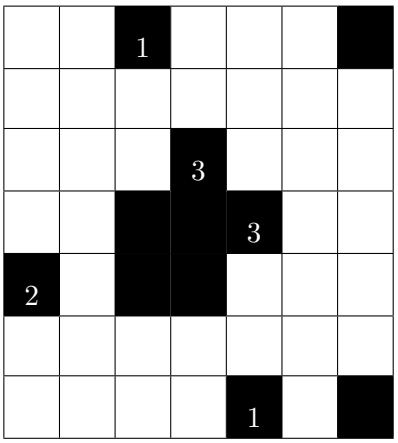
```
-,-,-,-,-,*,-
-,X,X,*,-,1,-
-,-,*,4,*,X,-
-,*,X,*,3,*,-
```

```
-,X,-,1,-,-,*
-,2,*,-,1,X,-
-,-,-,-,*,-,-
```

Para que puedas probar el funcionamiento de tu tarea, se proveerán 3 archivos que representan 3 tableros. Ten en consideración que el funcionamiento de tu tarea será con tableros distintos a los que se dan para probar. Cada uno de los 3 tableros tiene un nivel de dificultad distinto: fácil, medio y difícil. Los tableros y el nombre del archivo respectivo se muestran a continuación:

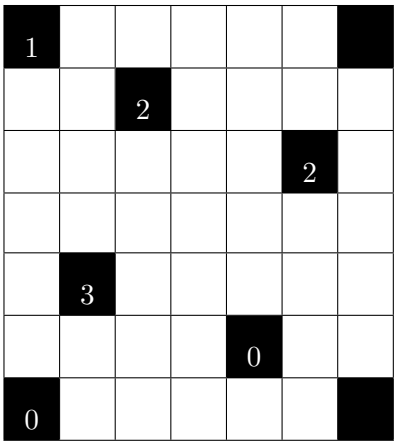
Tableros de prueba: Fácil

Archivo `'facil.txt'`



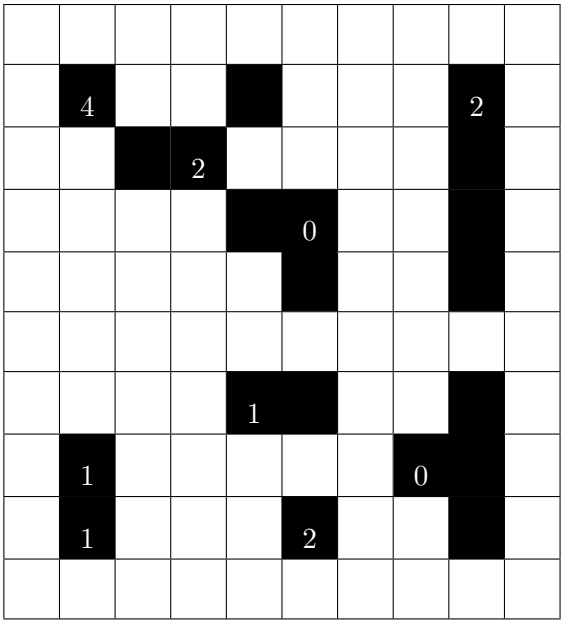
Tableros de prueba: Difícil

Archivo `'dificil.txt'`



Tableros de prueba: Medio

Archivo `'medio.txt'`



Recursión

En esta tarea también se evaluará contenidos de recursión. Para esto, es necesario que el método `resolver_tablero` sea recursivo. Si se realiza de manera iterativa, no se otorgará puntaje, aunque esté bien implementada.
Si estimas conveniente, el resto de los métodos también podrían ser recursivos, pero no es requisito.

Funcionalidades de tu tarea

Tu tarea debe tener las siguientes funcionalidades:

1. Saludar al usuario y preguntarle si quiere jugar una nueva partida o cargar una partida guardada previamente. En caso de jugar una nueva partida, se le debe pedir el nivel de dificultad del tablero que quiere jugar. Si el usuario quiere cargar una partida cargada previamente, se debe pedir el nombre del archivo. Puedes asumir que el archivo siempre se encontrará en la misma carpeta que tu programa.

2. Mostrar el tablero en consola y permitir que el usuario realice alguna de las siguientes opciones:

   - Realizar una jugada: el usuario pone una ampolleta en una celda. Tu programa deberá verificar que la celda esté dentro de las dimensiones del tablero y que la ampolleta no viole ninguna de las reglas del juego.
   - Resolver tablero: el usuario pide que se entregue la solución del tablero original que está jugando y ésta se muestra en la consola. Con esto el programa termina.
   - Eliminar una ampolleta: el usuario da la celda donde quiere eliminar una ampolleta. Si la celda contiene una ampolleta, se borrará.
   - "Prender luces": imprime en pantalla el tablero marcando, de alguna manera, todas las celdas que están iluminadas.
   - Guardar partida actual: el usuario puede querer guardar el estado actual del tablero en un archivo txt.
   - Salir del juego: el usuario no termina de completar el tablero y tampoco quiere guardar el estado actual. Con esto tu programa deberá terminar.

3. Llevar registro de la cantidad de jugadas correctas que el usuario ha realizado, junto con la cantidad de jugadas incorrectas (si viola alguna condición de las ampolletas). Desplegar estos datos después de cada jugada del usuario.

4. Felicitar al usuario cuando ha logrado completar el tablero con una solución correcta (que cumple con todas las restricciones). Esta felicitación no se debe dar cuando el usuario ha pedido mostrar la solución del tablero

Nota: no es necesario tener el tablero Akari resuelto para poder jugar, solo se debe resolver en caso que el usuario lo pida

Bonus

Tendrás la opción de tener nota máxima 7.5 en tu tarea. Para lograr esto, tu tarea debe cumplir lo siguiente:

- Tener implementada la totalidad de la tarea según lo descrito en la sección anterior
- Realizar alguna mejora en el desempeño del algoritmo que resuelve el tablero, considerando el tiempo que le tome al algoritmo converger en una solución.
- Dejar al comienzo de tu tarea un comentario diciendo que estás optando al Bonus, además de una justificación de cómo mejoraste el desempeño del algoritmo.

El ayudante probará el tiempo en que el algoritmo resuelve el tablero utilizando el tablero `bonus.txt`. Por último, si obtienes este bonus, debes tener en consideración que la nota final del curso no puede sobrepasar la nota 7.0.

Entrega

Debes guardar tu tarea en **un** archivo con el formato `tarea03_rut.py`, donde debes reemplazar `rut` con tu RUT. Por ejemplo si tu rut es 12.345.678-9 el nombre de la tarea debería ser `tarea03_123456789.py`. La entrega se realiza en el buzón electrónico, disponible en la página web del curso: `https://puc.classter.net` hasta el 17 de noviembre a las 23:59 hrs.

Ejemplo

A continuación se muestra una posible interacción con el usuario:

```
***** Bienvenido a Akari *****
Que quieres hacer?
1. Jugar una nueva partida
2. Jugar una partida guardada anteriormente
>>> 1
Que dificultad quiere jugar ?
1. Facil
2. Medio
3. Dificil
>>> 1
****************************
  -  -  1  -  -  -  X
  -  -  -  -  -  -  -
  -  -  -  3  -  -  X
  -  -  X  X  3  -  -
  2  -  X  X  -  -  -
  -  -  -  -  -  -  -
  -  -  -  -  1  -  X

Lleva 0 intentos correctos y 0 intentos incorrectos
Que quiere hacer?
1. Realizar jugada
2. Resolver tablero
3. Eliminar una ampolleta
4. Prender luces
5. Guardar partida actual
6. Salir del juego
>>> 2
Resolviendo el tablero...

  -  *  1  -  -  -  X
  -  -  -  *  -  -  -
  -  -  *  3  *  -  X
  *  -  X  X  3  *  -
```

```
2  -  X  X  *  -  -
*  -  -  -  -  -  -
-  -  -  *  1  -  X
```