



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada (II/2016)
Tarea 2

1. Objetivos

- Aplicar conceptos y nociones de estructuras de datos para el correcto modelamiento de un problema.

2. Introducción

¡Tu juego fue todo un éxito! Sin embargo, luego de ser humillantemente derrotado en la liga Kalos, nuestro héroe Bastián pasó demasiado tiempo encerrado en su PC y cuando volvió solo hablaba de pinturillo¹ y pokémon go. En cuanto escuchaste esto te diste cuenta de que fueron horas perdidas de programación, ya que tú querías jugar un juego que fuera algo más desafiante para matar todo ese tiempo libre que posees, el juego que en realidad querías hacer era el Go².

Así que te pones manos a la obra para lograr tu sueño de ser el mejor *entrenador pokémon* jugador de Go del mundo.

3. Problema

En esta tarea, deberás modelar las distintas funcionalidades del juego **"Go"** para crear un cliente de juego completamente funcional capaz de leer y guardar partidas en archivos con formato sfg. Su cliente debe forzar todas las reglas del juego explicadas más adelante. Lo anterior **debe** ser logrado utilizando conceptos y nociones de **estructuras de datos**.

4. GO

GO (Baduk/Weiqi) es un juego de dos jugadores que se juega en un tablero de 19x19 líneas.³ Un jugador juega con las fichas negras y el adversario juega con las de color blanco. Los jugadores se turnan poniendo *piedras* en las intersecciones del tablero.

4.1. Poniendo y capturando piedras

Una vez que una piedra ha sido puesta no puede ser movida, sin embargo las piedras pueden ser capturadas.

¹ Ni idea de qué es eso

² En parte porque quieres dejar de tener esos sueños raros con programones

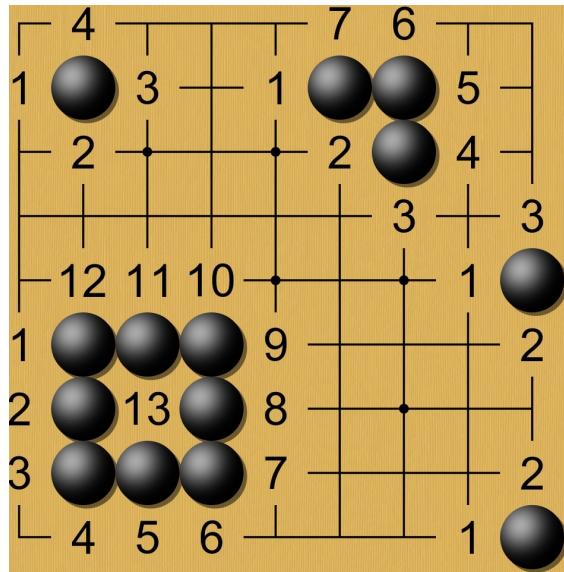
³ La mayoría de los ejemplos se dan en un tablero de 9x9 líneas por un tema de espacio, sin embargo los conceptos son generalizables

Una piedra puede ser puesta en cualquier intersección libre del tablero, siempre y cuando no se trate de una jugada suicida o de una violación a la regla de *ko* (explicada más adelante). Es decir una piedra (o grupo) de un jugador no puede ser capturada (o capturado) como resultado directo de una jugada de ese mismo jugador⁴.

Debe asegurarse de no permitir a un jugador hacer una jugada ilegal.

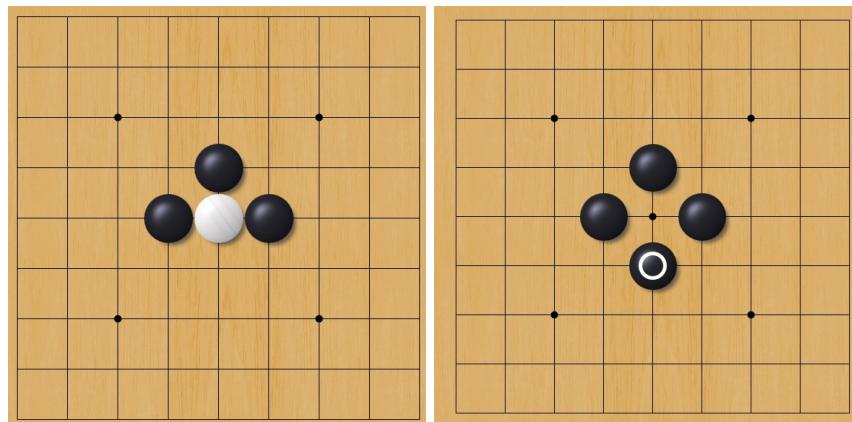
4.1.1. Libertades

Las intersecciones que están junto a una piedra o grupo de piedras son sus *libertades*. Dos (o más) piedras que están contiguas forman un grupo y por lo tanto comparten libertades. A continuación se puede ver un ejemplo de como se cuentan las libertades de una piedra o grupo de piedras (notar que las diagonales no cuentan y que las intersecciones interiores sí lo hacen):



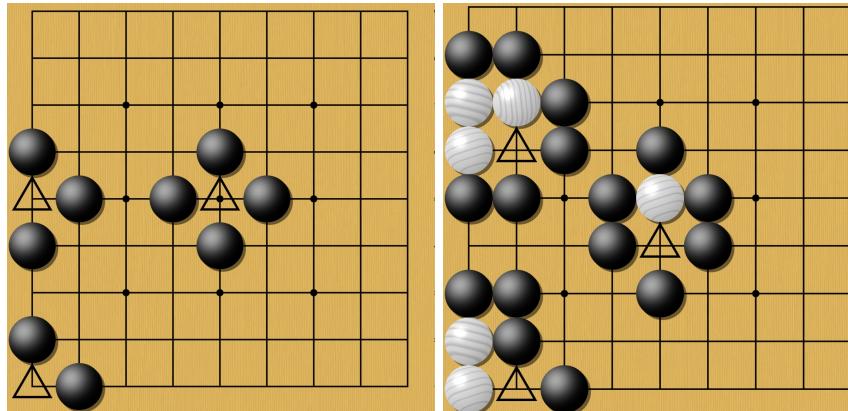
4.1.2. Captura

Las piedras son capturadas al tapar todas sus libertades (rodeándolas completamente). Las piedras capturadas son tomadas como *prisioneros* por el jugador que las capturó y son sacadas del tablero.

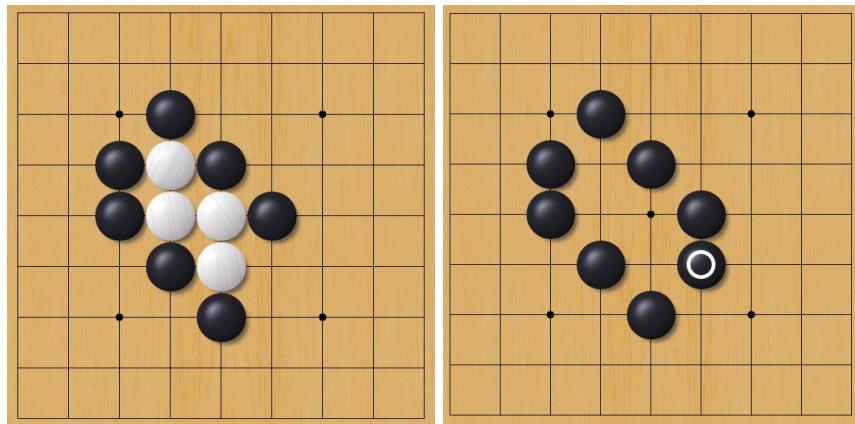


⁴ Pues sería una jugada suicida...

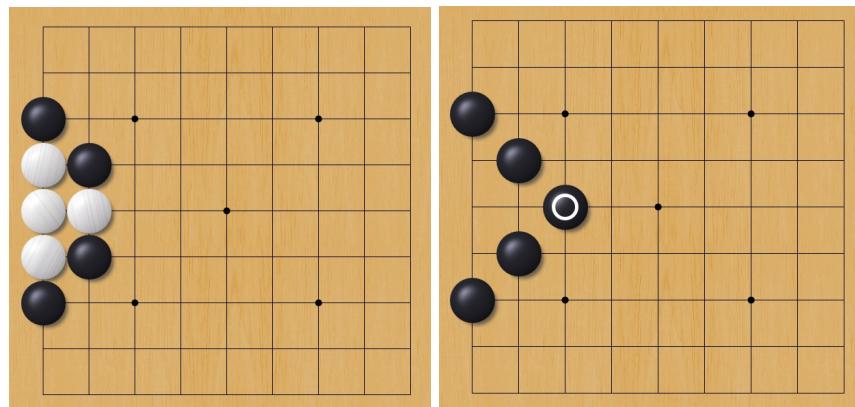
Las siguientes imágenes muestran posiciones donde el jugador blanco **no** podría jugar de ser su turno (Las posiciones inválidas están marcadas con \triangle):



Un grupo de piedras puede ser capturado si se rodea completamente.

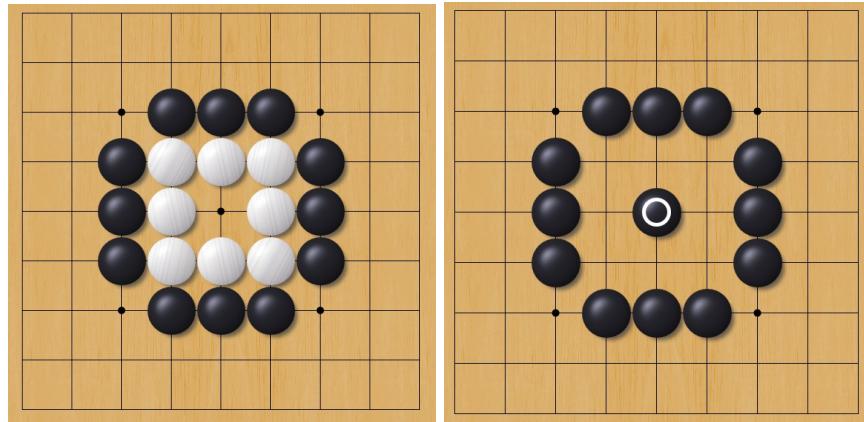


Las piedras que estén en el borde solo deben ser rodeadas en el espacio del tablero para ser capturadas.



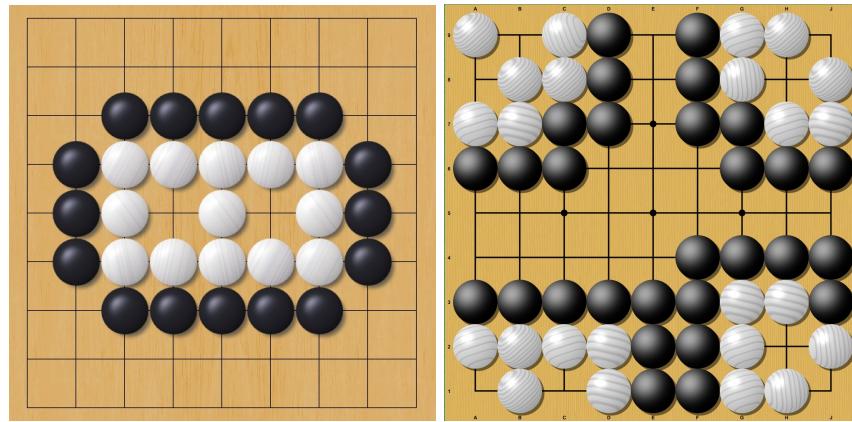
Debe asegurarse de quitar automáticamente las piedras del tablero en cuanto se haga una jugada que implique que estas resulten capturadas.

Simplemente rodear un grupo por fuera no es suficiente para capturarlo. Deben taparse todas las libertades del grupo. (Notar que esto es una jugada válida ya que primero han de removese del tablero las piedras capturadas y luego chequear que no corresponda a una jugada suicida).

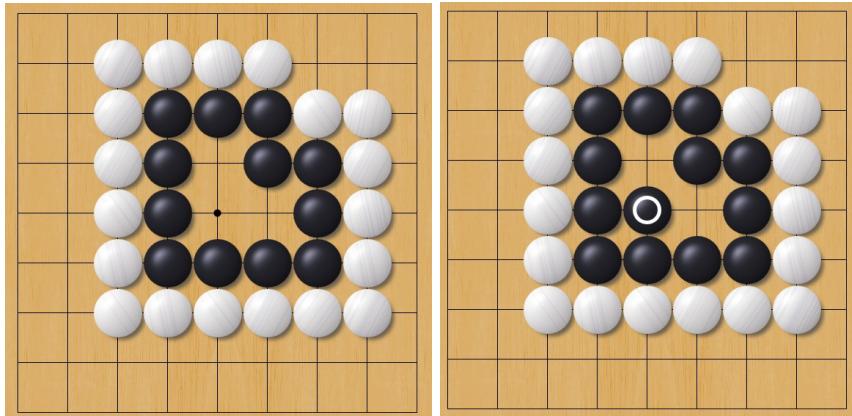


Esto implica que un grupo solo está verdaderamente a salvo si tiene por lo menos dos 'ojos'. Esto es así puesto que si el otro jugador intentase capturarlos, aun si lo rodea completamente, el grupo tendrá dos libertades (las interiores) y el otro jugador no podrá nunca tapar dichas libertades pues ambas posiciones corresponden a jugadas suicidas y por lo tanto son ilegales.

Por ejemplo, es imposible capturar los siguientes grupos de Blanco (A menos que Blanco tontamente rellene uno de los ojos).



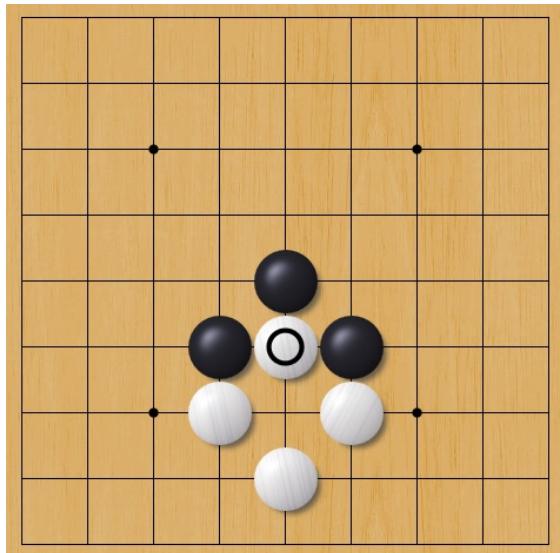
En el siguiente ejemplo, para proteger a su grupo de ser capturado, Negro puede formar dos ojos:



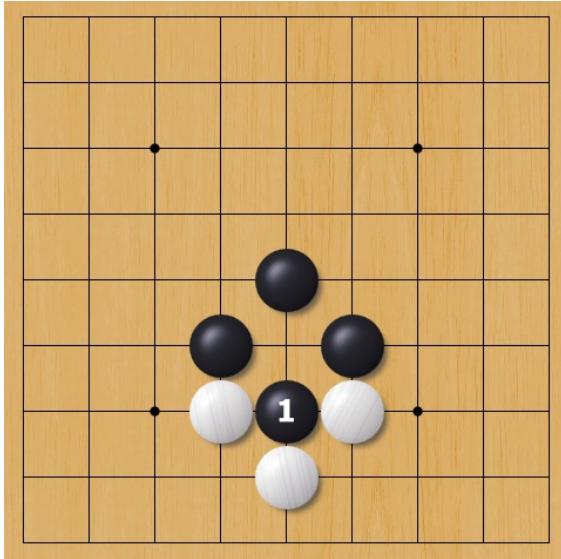
4.2. Ko

Hay una regla especial llamada **Regla de Ko** la cual evita ciclos infinitos de captura. La regla es simple, no puedes hacer un ciclo. Es decir no está permitido repetir el estado del tablero al jugar una piedra.

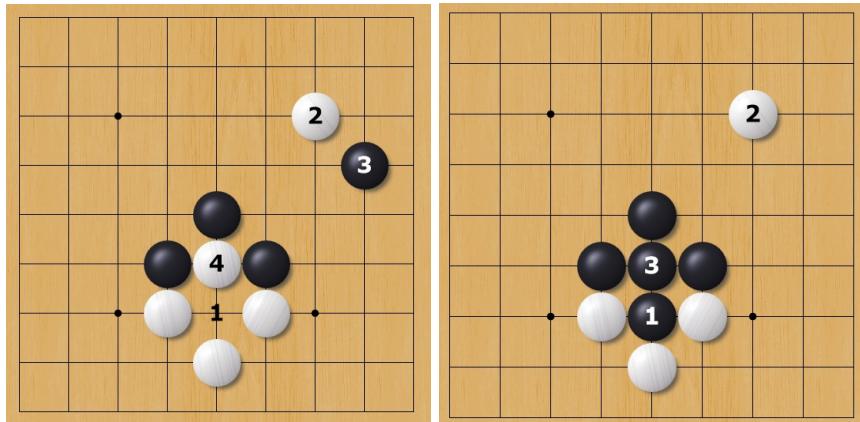
Por ejemplo, supongamos este es el estado del juego:



Ahora supongamos que Negro captura la piedra que blanco acaba de jugar:



Blanco podría volver a capturar, sin embargo si lo hace inmediatamente estaría violando la Regla de Ko. Por lo tanto Blanco tiene que jugar en otra posición y no puede capturar el ko en este turno. Luego de que Blanco juegue en otra posición (2) Negro puede jugar en cualquier posición válida. Si Negro decide conectar sus piedras, termina el ko. En caso contrario, Blanco podría volver a capturar con (4), y entonces Negro no podría volver a capturar de inmediato y tiene que jugar en otra posición.

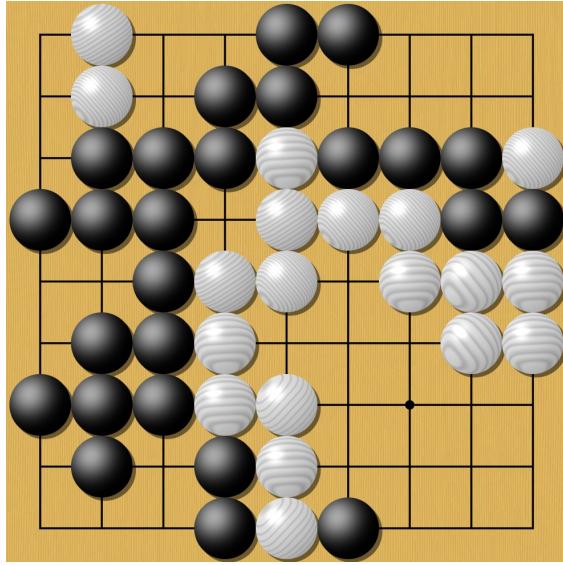


4.3. Término del juego y puntaje

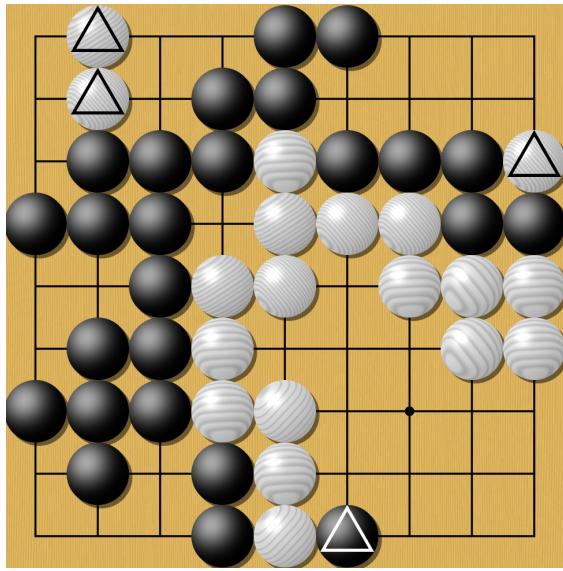
Cuando un jugador cree que no tiene jugadas útiles, pasa. Cuando los dos jugadores pasan consecutivamente entonces el juego se termina y se determina el ganador contando el *territorio* que tiene cada jugador añadiendo esto al número de prisioneros que han capturado.

4.3.1. Quitando las piedras muertas

Antes de contar el *territorio* se deben remover del tablero las piedras restantes que ambos jugadores consideran muertas. Por ejemplo, consideremos la siguiente situación donde ambos jugadores acaban de pasar (es decir que ambos piensan que no tienen jugadas útiles):



Ahora los jugadores deben ponerse de acuerdo respecto a qué piedras de las restantes están muertas. Las piedras se consideran muertas si no tienen la posibilidad de hacer un grupo vivo o de conectarse a un grupo vivo. En este caso corresponde a las marcadas con Δ .



Estas piedras se remueven entonces del tablero y se consideran capturadas. Una vez hecho esto, se puede pasar a contar el territorio.

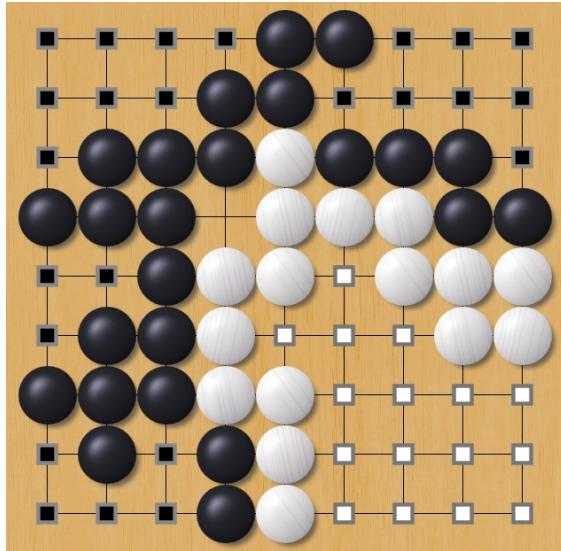
Llegada a esta parte del juego, deben marcarse las piedras/grupos que serán eliminadas al hacer click en ellas (deberá bastar hacer click en una sola piedra de un grupo para marcar el grupo completo) para luego moverse haciendo click en el botón count. Debe procurar no permitir eliminar grupos que tengan 2 o más ojos.

4.3.2. Territorio

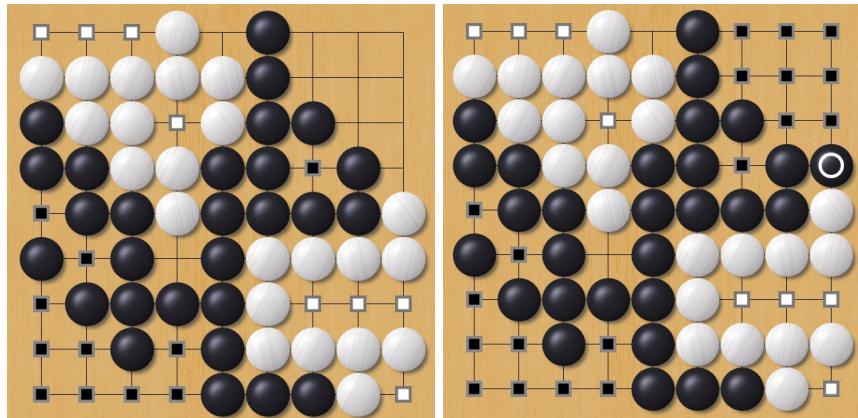
El territorio es la principal fuente de puntos en el juego, y conseguir la mayor cantidad de territorio es el objetivo de este.

El territorio es una parte del tablero que está rodeada por piedras pertenecientes a grupos vivos de un jugador y dentro del cual no es posible que el oponente haga un grupo vivo. Notar que solo se cuentan las intersecciones vacías controladas por un jugador y **no** las piedras que delimitan el territorio.

Siguiendo con el ejemplo, Negro tiene 24 puntos de territorio y blanco tiene 16 puntos de territorio:



Al contar el puntaje el territorio tiene que estar completamente rodeado para poder ser contado.



Tras hacer click en el botón count (y después de remover del tablero las piedras muertas) debe contarse automáticamente el territorio de cada jugador, marcando cada intersección de este con un cuadrado del color correspondiente al igual que en el ejemplo.

4.3.3. Komi

Debido a que Negro parte jugando, y que esto constituye una ventaja es que el jugador Blanco tiene un bonus especial de puntaje llamado komi.

Así, el puntaje del jugador Negro será:

territorio + prisioneros

Mientras que el puntaje del jugador Blanco será:

territorio + prisioneros + komi

El valor del komi Corresponde normalmente a 6.5 puntos y debe usar este valor al crear sus partidas. Sin embargo debe considerar que puede especificarse otro valor en los archivos sgf.

4.3.4. Resultado

El resultado de la partida se expresa de la siguiente manera:

Ganador+PorCuantoGanó

Donde **Ganador** será W si el ganador fue el jugador Blanco (White) y B si el ganador fue el jugador Negro (Black). Mientras que **PorCuantoGanó** indicará la diferencia de puntaje entre el ganador y el perdedor.

Por ejemplo si al final de la partida Negro termina con X puntos y Blanco con X+15.5 puntos, entonces el resultado será:

W + 15,5

En caso de que algún jugador renuncie antes del final de la partida entonces **PorCuantoGanó** se anotará como **res**

Por ejemplo si Blanco se rinde entonces el resultado será:

B + res

5. Archivos

Tu programa debe ser capaz de leer y escribir archivos con extensión SGF (Smart Game Format). El archivo contiene información relevante, como el tamaño del tablero y el valor del Komi. Además posee todas las jugadas que se han realizado durante la partida en forma de árbol, esto porque durante el transcurso del juego los jugadores pueden crear distintas líneas de juego a partir de un estado en común (esto se explicará mejor en el punto siguiente).

Pueden leer acerca de las especificaciones del formato sgf en esta página.

5.1. Formato Archivos

El formato de los archivos es el siguiente:

```
(;GM[1]FF[4]CA[UTF-8]SZ[19]KM[6.50] (;B[dd] ) ;W[dd] ;B[pq] ;W[dp] ;B[oc] ;W[po]
;B[qo] ;W[qn] ;B[qp] ;W[pm] ;B[nq] ;W[qi] ) ;B[fc] ;W[cf] ;B[db] ;W[cc] ;B[ic]))
```

El significado de cada uno de estos elementos es el siguiente:

- GM[]: representa a qué juego pertenece el tablero, en nuestro caso "GO"(1).
- FF[]: Es la versión del formato (4 para todos los casos).

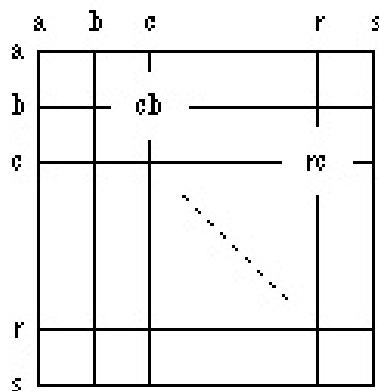
- CA[]: Es el encoding del archivo (en general UTF-8).
- SZ[]: Representa el tamaño del tablero en líneas (19).
- KM[]: Es el valor del Komi.

Esto nos da información de la partida y algunos metadatos. Esta información debe estar en los archivos que el programa genera.⁵

Posteriormente se encuentran todas las jugadas de la siguiente forma:

- B[xy]: Negro jugó una pieza en la posición (x, y).
- W[xy]: Blanco jugó una pieza en la posición (x, y).

El sistema de coordenadas es el representado por la siguiente imagen:



Los archivos sgf pueden contener más información, por ejemplo:

- EV[]: Provee el nombre del evento (p. ej. un torneo).
- DT[]: La fecha (en formato ISO) de cuando se jugó la partida. En caso de que la partida dure más de un día se separarán por una coma.
- PB[]: Nombre del jugador negro.
- BR[]: Rank del jugador negro.
- PW[]: Nombre del jugador blanco.
- WR[]: Rank del jugador blanco.
- RE[]: Resultado de la partida.
- SO[]: Nombre de la fuente (p. ej. libro, revista, página de internet,...)

No es necesario que el programa escriba esta información, sin embargo no debe tener problemas al trabajar con archivos que sí la contengan.

⁵ Al abrir un archivo este puede no contener esta información, en ese caso deberá asumir: GM[1], FF[4], CA[UTF-8],SZ[19], KM[6.5]

5.2. Variaciones y estructura del archivo

Este formato permite codificar variaciones. Cada variación comienza y termina con paréntesis. Por ejemplo:

(;B[aa] ;W[ab] ;B[ac])

Representa una variación de tres jugadas. Mientras que:

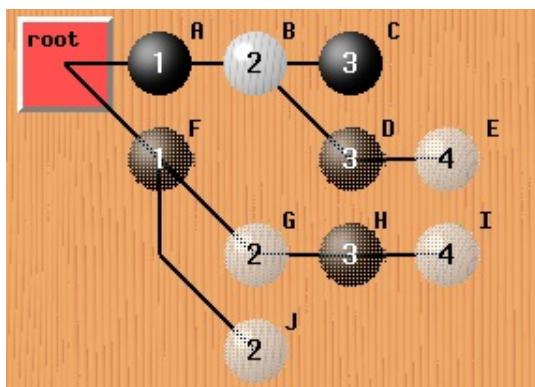
(;B[dd])(;B[qd] ;W[dd] ;B[pq])

Muestra dos variaciones, la primera de una jugada y la segunda de tres.

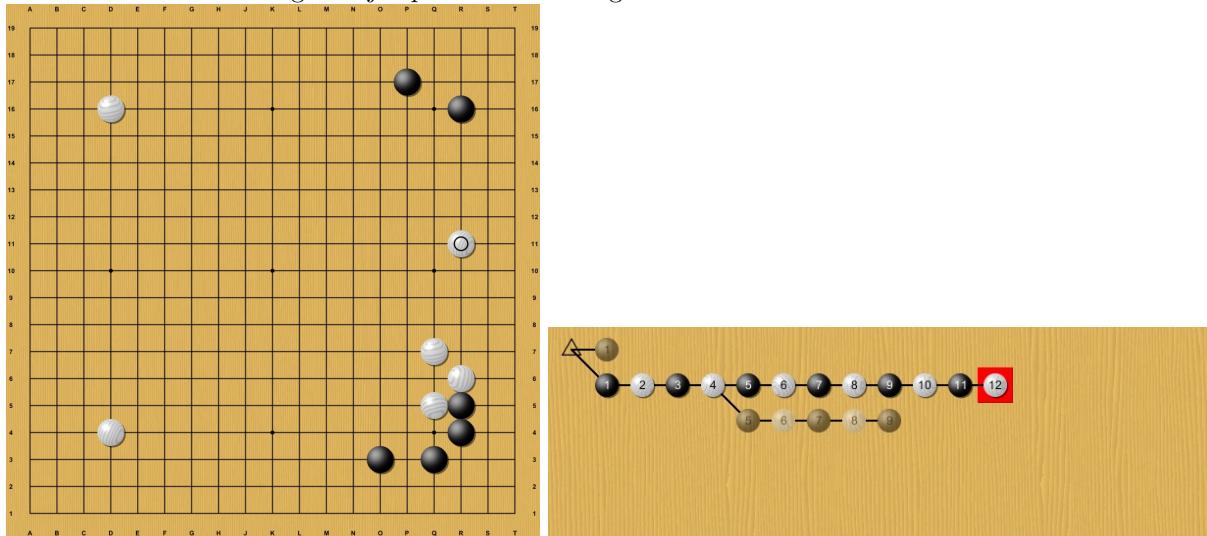
Como ejemplo de la estructura de la información, un archivo de la siguiente forma (C[]: representa un comentario⁶):

(;FF[4]C[root](;C[a];C[b](;C[c])(;C[d];C[e]))(;C[f](;C[g];C[h];C[i])(;C[j])))

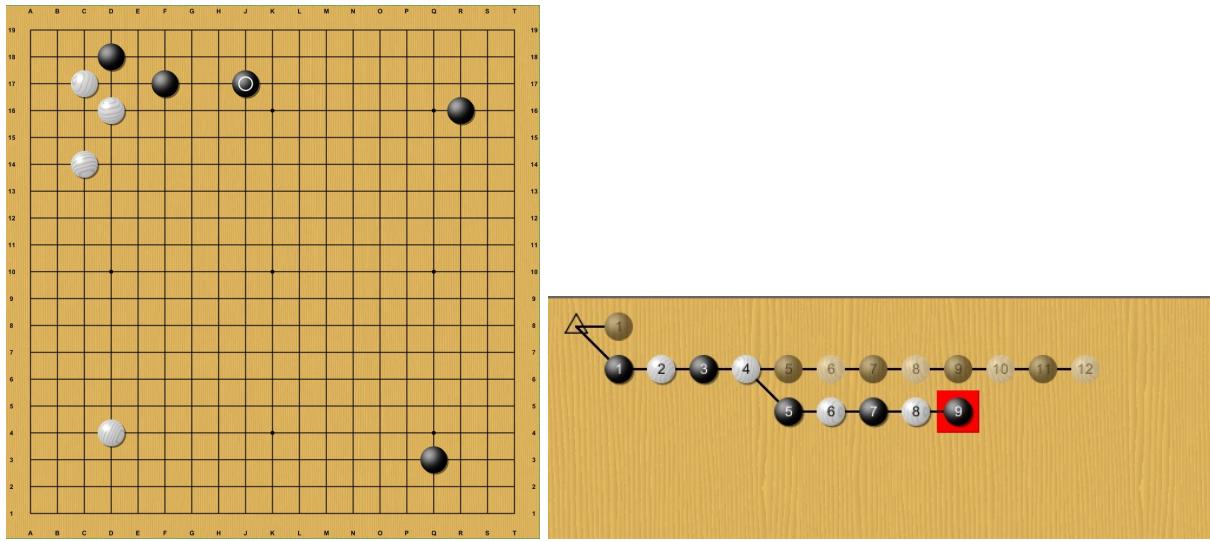
Se vería así:



Uno de los archivos sgf de ejemplo codifica las siguientes variaciones:



⁶ No es necesario que manejen la propiedad C[] en su tarea



Deben mostrarse las variaciones en la interfaz de forma similar a la mostrada anteriormente. Además, al hacer click en un nodo del árbol de variaciones, debe mostrarse en el tablero el estado correspondiente a ese nodo, permitiendo continuar el juego desde ese punto en adelante formando una **nueva variación** cuando corresponda (desde el punto en que las jugadas diverjan).

Debe poder generar archivos en este formato a partir de un juego nuevo, o bien habiendo cargado inicialmente un archivo sgf y habiéndolo modificado.

Para más información pueden ir acá

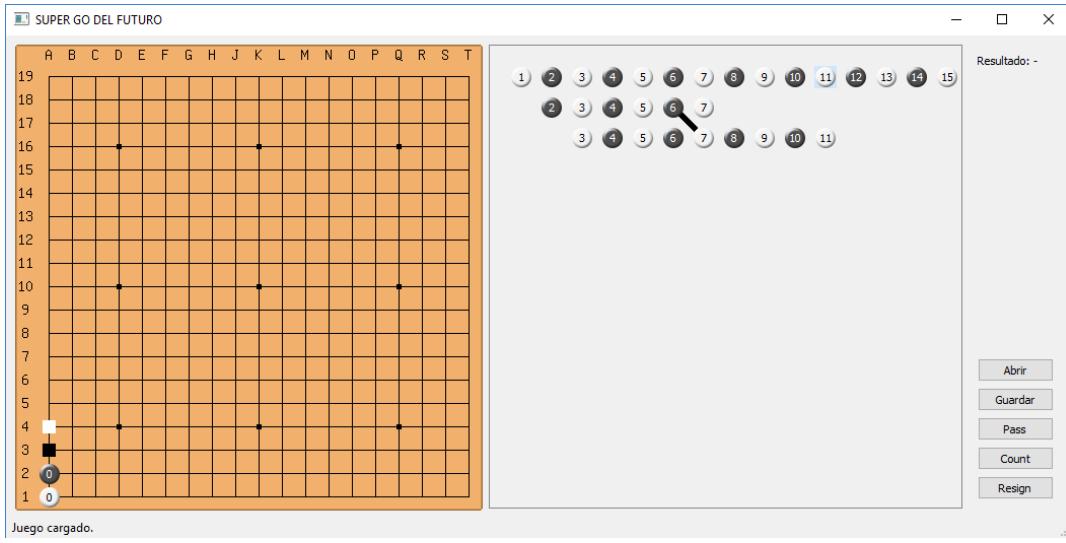
6. Probar el juego

Les recomendamos probar este juego para acostumbrarse al desarrollo de la tarea:

- En OGS (OnlineGOServer) pueden encontrar un tutorial así como oponentes para jugar (funciona desde el navegador).
- En KGS (KGSGOserver) pueden encontrar un cliente que les permite tanto conectarse a un servidor para jugar como abrir/crear/editar archivos sgf.

7. Interfaz

La interfaz sobre la cual usted va a trabajar es la siguiente:



La interfaz cuenta con el tablero, el árbol y los botones. El tablero utiliza un sistema de coordenadas de letras y números. El árbol utiliza un sistema de coordenadas (i,j) donde 0,0 es la coordenada del extremo superior izquierdo.

La interfaz está representada mediante la clase `MainWindow` en el módulo `gui`. En esta clase se encuentran todos los métodos que necesita para interactuar con la interfaz.

7.1. MainWindow

Los métodos de esta clase se pueden dividir en dos grandes grupos: **Callbacks** y **directos**.

Los métodos de tipo callback son métodos que serán ejecutados cuando el usuario interactue con la interfaz (principalmente clicks en lugares clave de la interfaz). Estos métodos son:

`on_piece_click(self, letter, y)`: Este método es llamado cada vez que se clickea una pieza en el tablero y entrega su letra y su número.

`on_point_click(self, i, j)`: Este método es llamado cada vez que se clickea un nodo en el árbol y entrega sus coordenadas.

`on_open_click(self, path)`: Este método es llamado cuando un usuario elige un archivo para cargarlo en la interfaz y entrega el path del archivo.

`on_save_click(self, path)`: Este método es llamado cuando un usuario elige un archivo para guardar el estado actual del árbol y entrega el path del archivo.

`on_pass_click(self)`: Este método es llamado cada vez que se hace click en el botón de pasar.

`on_count_click(self)`: Este método es llamado cada vez que se hace click en el botón de contar territorios.

`on_resign_click(self)`: Este método es llamado cada vez que se hace click en el botón de rendirse.

Los métodos directos sirven para interactuar con la interfaz, principalmente agregando piezas al tablero, nodos y cambiando los mensajes y títulos de la interfaz.

`add_piece(self, letter, y, text, color)`: Este método agrega una ficha al tablero, recibe la letra, la coordenada y, el número sobre la ficha y el color de la ficha en inglés `white` o `black`.

`add_square(self, letter, y, color)`: Este método agrega un cuadrado al tablero, recibe la letra, la coordenada y y el color del cuadrado en inglés `white` o `black`,

`remove_piece(self, letter, y)`: Este método remueve el cuadrado o pieza en la posición especificada. Si no se encuentra esta pieza se imprime un error en consola.

`add_point(self, i, j, text, color)`: Este método agrega un punto al árbol en la coordinada específica.

`remove_point(self, i, j)`: Este método remueve el punto del árbol en la posición especificada. Si no se encuentra este punto se imprime un error en consola.

`add_line(self, cord_1, cord_2, color)`: Este método agrega una linea del color especificado entre los puntos ubicados en las coordenadas especificadas. Los parámetros de coordenadas reciben objetos que implementen `__getitem__` (como listas, tuplas, sets, etc.). Si su estructura no implementa este método, entonces puede pasar los parámetros como `add_line((cord1_i, cord1_j), (cord2_i, cord2_j), color)`. Cabe decir que esta es la **única** vez en que estará bien justificado instanciar estructuras nativas de python, aunque se recomienda que sus estructuras implementen `__getitem__`.

`remove_line(self, cord_1, cord_2)`: Este método remueve una linea entre los puntos ubicados en las coordenadas especificadas. Si no se encuentra esta linea se imprime un error en consola.

`set_result(self, text)`: Este método cambia el texto después de `Resultado`:

`show_message(self, text)`: Este método cambia el en la barra inferior de la interfaz.

`setWindowTitle(self, text)`: Este método cambia el título de la interfaz.

7.2. Módulo gui

En el módulo `gui` viene la clase `MainWindow` que usted debe extender y viene el método `run(window)` que recibe una instancia de `MainWindow` y la despliega, en el archivo `demo` usted podrá ver claramente como funciona cada método y el método `run(window)`.

8. Estados iguales

Es posible notar que a veces se llega al mismo estado del tablero pero mediante distintas variaciones. Cuando esto ocurra, se deberán conectar dichos nodos con una linea blanca usando el método `dado`. Esto debe ocurrir tanto si se trata de un archivo `sgf` que se está abriendo como si es una partida (con sus respectivas variaciones) creada en el cliente.

9. Dan Level

Para demostrarle al mundo tu gran habilidad y entendimiento tanto del juego como de las estructuras de datos, deberás implementar el juego **SIN USAR LAS SIGUIENTES ESTRUCTURAS DADAS POR PYTHON: DICIONARIOS, LISTAS NI CONJUNTOS**. Debes detallar en el `README` las especificaciones de cómo implementaste cada una de las estructuras utilizadas.

Si necesitas usar alguna función perteneciente a algún módulo de la librería estándar de Python que retorne alguna de las estructuras prohibidas, deberás desempaquetar los valores de la estructura mediante el operador `*` y luego usarlos en tu propia implementación. Por ejemplo, el siguiente código es válido:

```
class MiLista:  
    def __init__(self, *args):  
        for arg in args:  
            # do something  
            pass  
  
separa2 = "lorem ipsum dolor sit amet".split(" ")
```

```
lista_legal = MiLista(*separa2)
lista_legal.mi_append("consectetur")
```

Te recomendamos fuertemente que aproveches los métodos mágicos de Python (más información [aquí](#)). Por ejemplo, si quieras que tu implementación de listas sea iterable debes definir los métodos `__iter__` y `__next__`.

10. Bonus!

Esta tarea poseerá un bonus porcentual con respecto a la nota obtenida.

Se deberá implementar lo especificado en 4.3.1 (quitar piedras muertas al final de la partida) de forma automática además de manual. Es decir, una vez que los dos jugadores pasan de forma consecutiva, se deberán marcar automáticamente las piedras que serán eliminadas (manteniendo el comportamiento descrito anteriormente, se debe permitir marcar/desmarcar piedras haciendo click en ellas, donde deberá bastar un solo click para seleccionar un grupo completo y no debe permitirse seleccionar grupos que tengan dos o más ojos).

Cuando se haga click en el botón count deben removverse las piedras marcadas y luego se deberá contar el territorio que posee cada jugador.

11. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.5
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos PEP8
- Si no se encuentra especificado en el enunciado, asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje⁷ de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debe adjuntar un archivo `README.md` donde comente sus alcances y el funcionamiento de su sistema (i.e. manual de usuario) de forma concisa y clara.
- Cree un módulo para cada conjunto de clases. Divídala por las relaciones y los tipos que poseen en común. **Se descontará hasta un punto si se entrega la tarea en un solo módulo**⁸.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por sobre otro.

12. Entrega

- **Fecha/hora:** 9 de Septiembre - 23:59 hrs
- **Lugar:** GIT - Carpeta: Tareas/T02

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).

⁷ Hasta -5 décimas.

⁸ No agarre su código de un solo y lo divida en dos módulos, module su código de forma inteligente