Chia, Guido Giuliano (L.U.: 117825)

Frizzera, Ignacio (L.U.: 117815)

Comisión de Cursado N°9

Lógica para Ciencias de la Computación – 17/05/2019

Proyecto N°1

PROGRAMACIÓN EN EL LENGUAJE PROLOG

Contenido

[Introducción 2](#_Toc8677463)

[Uso del Sistema 3](#_Toc8677464)

[Ejecución del Programa 3](#_Toc8677465)

[Uso del Programa 3](#_Toc8677466)

[Interfaz del programa 4](#_Toc8677467)

[Colocación de una piedra en una posición invalida 5](#_Toc8677468)

[Finalización de partida y cálculo de puntajes 6](#_Toc8677469)

[Desarrollo del Programa 7](#_Toc8677470)

[checked(Player, Index) 7](#_Toc8677471)

[encerradoActual(Player, Index) 7](#_Toc8677472)

[noEncerrado(Index) 7](#_Toc8677473)

[emptyBoard(Board) 7](#_Toc8677474)

[goMove(+Board, +Player, +Pos, -RBoard) 7](#_Toc8677475)

[replace(?X, +XIndex, +Y, +Xs, -XsY) 8](#_Toc8677476)

[esValida(+Board, +Player, +Index) 8](#_Toc8677477)

[buscarEliminarVecinosEncerrrados(+Board, +Index, +Player, -NBoard) 8](#_Toc8677478)

[checkEncerrado(+Board, +Index, +Player, +Opponent, +Liberty, -NBoard) 8](#_Toc8677479)

[checkEncerradoCascara(+Board, +Player, +Opponent, +Liberty, -NBoard, +Index) 8](#_Toc8677480)

[estaEncerrado(+Board, +Player, +Opponent, +Liberty, +Index) 8](#_Toc8677481)

[getVecinos(+Index, -IndexesVecinos) 8](#_Toc8677482)

[vecinoEstaEncerrado(+Board, +Player, +Opponent, +Liberty, +ValuesVecinos, +IndexVecinos) 8](#_Toc8677483)

[noEstaEncerrado(+Board, +Player, +Opponent, +Index) 9](#_Toc8677484)

[getValueListOnBoard(+Board, +Indexes, -Values) 9](#_Toc8677485)

[getValueOnBoard(+Board, +Index, -Value) 9](#_Toc8677486)

[getValueLista(+Columna, +Lista, -Value) 9](#_Toc8677487)

[getListaIndex(+Index, +Board, -Lista) 9](#_Toc8677488)

[eliminarEncerradosActuales(+Board, -NBoard) 9](#_Toc8677489)

[winPoints(+Board, -PWhite, -PBlack) 9](#_Toc8677490)

[calcularPuntos(+Board, +Player, +Opponent, +Liberty, -P) 9](#_Toc8677491)

[calcularPuntosAux(Board, Player, Opponent, Liberty, Index) 9](#_Toc8677492)

[checkEncerradoSinUncheck(+Board, +Player, +Opponent, +Liberty, +Index) 9](#_Toc8677493)

[getNext(+Index, -NextIndex) 10](#_Toc8677494)

[checkedToEncerrado 10](#_Toc8677495)

[checkedToNoEncerrado 10](#_Toc8677496)

[Conclusión 11](#_Toc8677497)

# Introducción

En el presente informe se detallarán todos los aspectos relevantes asociados al desarrollo del Proyecto N°1 de la asignatura *Lógica para Ciencias de la Computación* de la Universidad Nacional del Sur para la comisión del Primer Cuatrimestre del año 2019.

Para su resolución se debió implementar el juego GO: un juego de mesa para dos jugadores que consiste de un tablero de 19 líneas horizontales (filas) por 19 líneas verticales (columnas) y piezas llamadas piedras (fichas) de color blanco y negro.

El instructivo de cómo se juega se encuentra en el archivo PDF del enunciado del proyecto.

El desarrollo del programa se dividió en dos partes:

* **Interfaz gráfica**: está consiste de una interfaz web, la cual fue implementada mediante tres lenguajes de desarrollo web: HTML, CSS y JavaScript. Esta interfaz fue desarrollada en su mayor parte por la catedra, excepto un par de detalles que fueron modificados.
* **Lógica del juego**: fue implementada utilizando el lenguaje Prolog. Esto incluye todos los aspectos relacionados al funcionamiento de GO. Esta lógica correrá mediante un servidor, el cual será un servidor web Pengines.

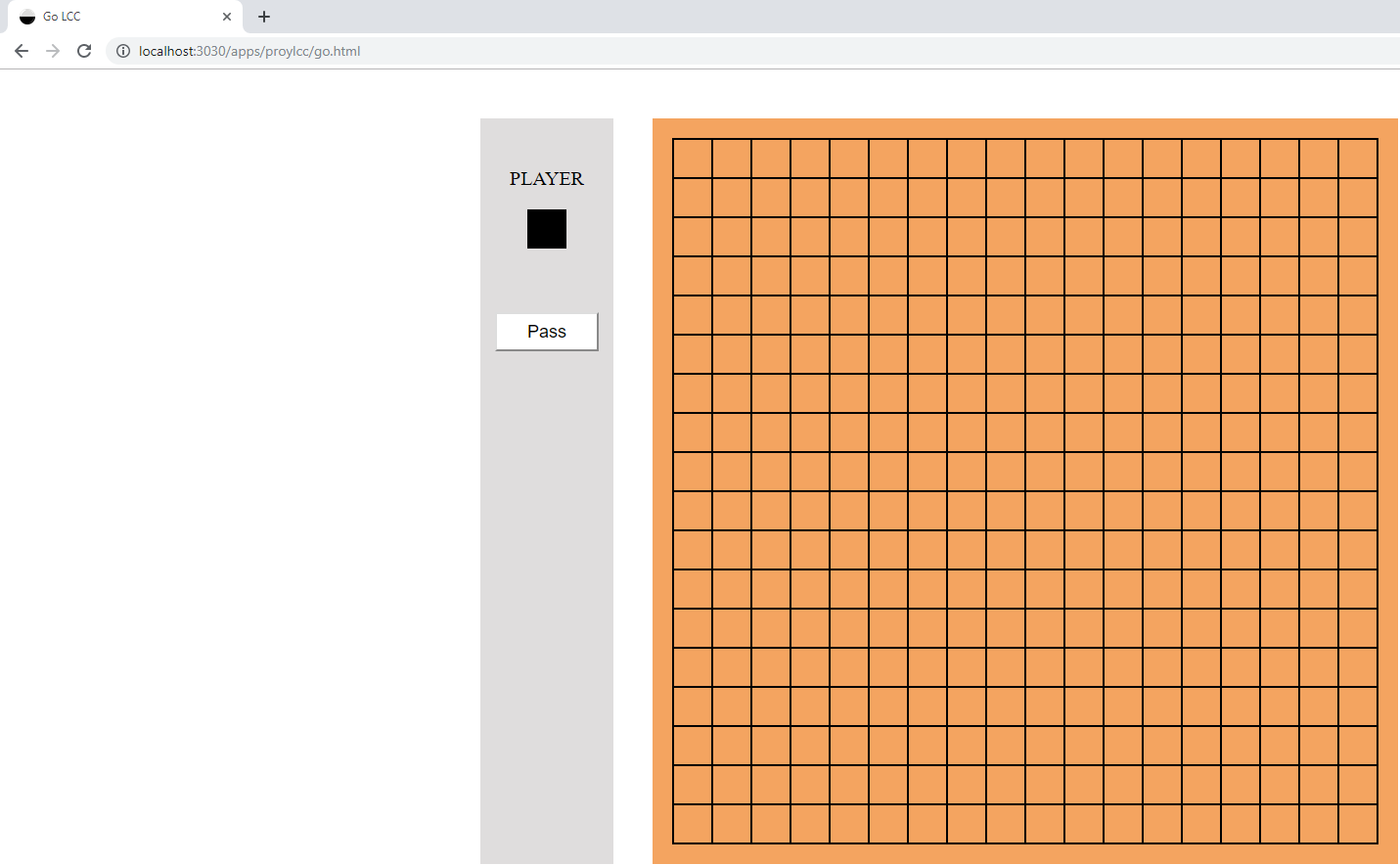
# Uso del Sistema

## Ejecución del Programa

Para comenzar, se debe tener instalado el programa *SWI-Prolog*, como también la implementación del servidor *Pengines*. Hecho esto se deberá ejecutar el archivo “run.pl” lo que levantara el servidor a utilizar, el cual escuchará en el puerto 3030 del localhost.

Una vez realizado lo mencionado anteriormente se deberá abrir en un buscador la siguiente url: [http://localhost:3030/apps/proylcc/go.html](http://www.google.com/url?q=http%3A%2F%2Flocalhost%3A3030%2Fapps%2Fproylcc%2Fgo.html&sa=D&sntz=1&usg=AFQjCNFwfTxl5KnymUdnclIRrW1dLCYrnw), lo cual comenzará la ejecución del juego.

Una vez inicializado el programa se presentará la siguiente venta:



Hecho esto se podrá comenzar a jugar.

## Uso del Programa

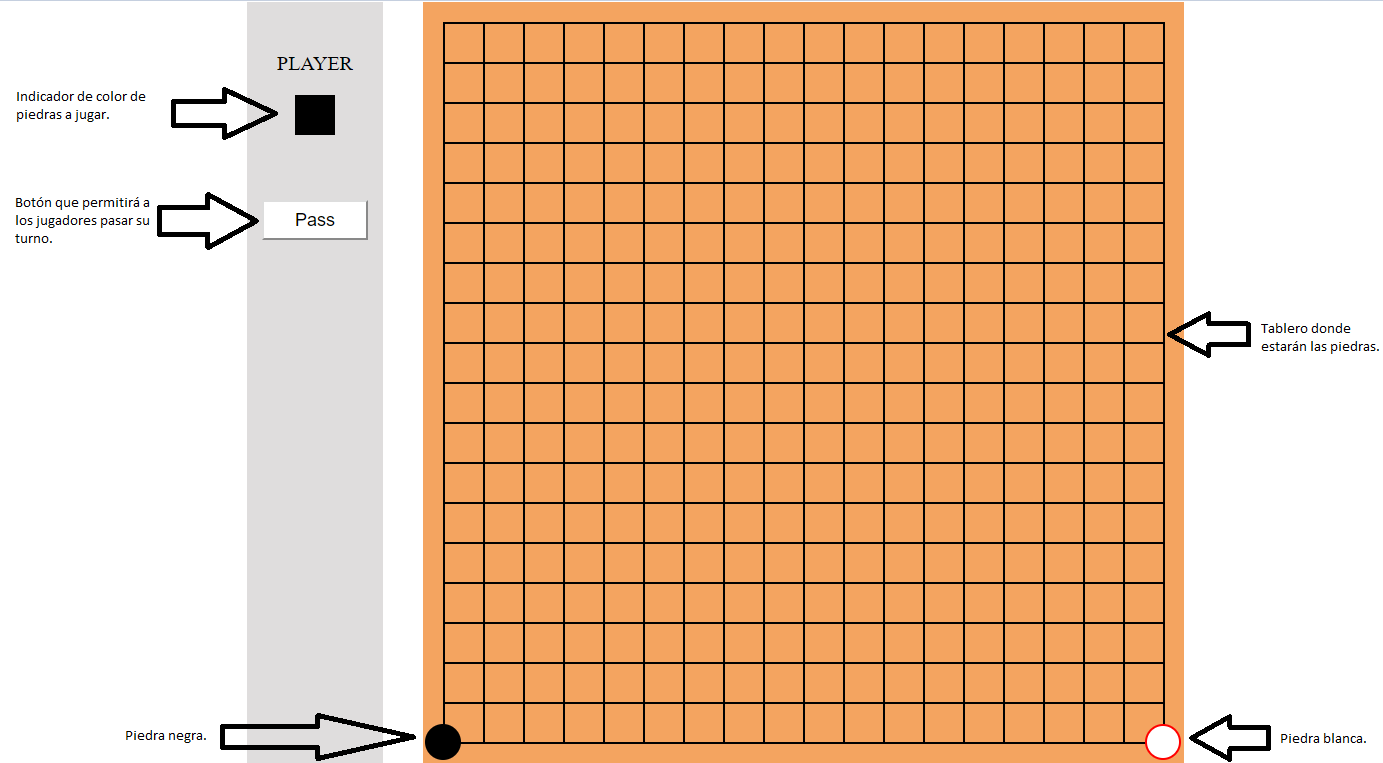
En principio, el programa presentará una interfaz la cual contendrá un tablero vacío donde se llevará acabo el desarrollo del juego. A medida que avancen la partida este tablero se comenzará a llenar de piedras ya sean de color blanco o negro.

Contará con un cartel que diga “PLAYER” y de bajo de este un cuadro, el cual indicará que color de piedras toca jugar.

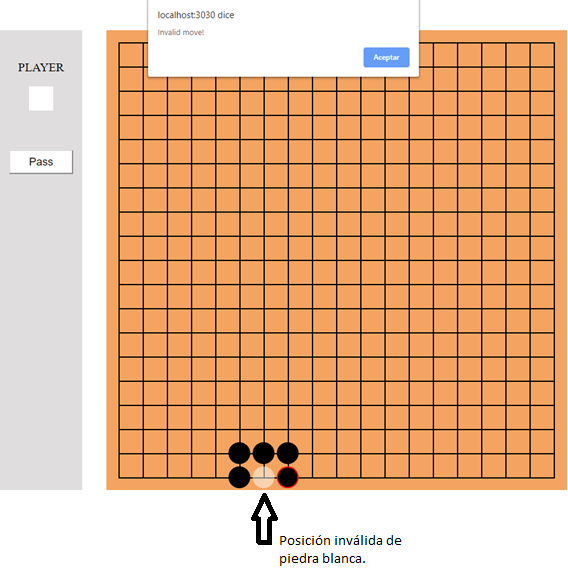
Para poder colocar una piedra se deberá hacer click en algún lugar del tablero que permita colocarla. En el caso de que el jugador no desee colocar una piedra, podrá hacer click en el botón “Pass” que le permitirá saltear su turno. En el caso de que tanto el jugador de piedras negras y el jugador de piedras blanca pasen de turno, la partida finalizará y se mostrará una notificación con los puntajes de cada jugador. Si se desea volver a jugar se deberá hacer click en el botón “Aceptar” de la notificación y así poder comenzar una nueva partida del juego.

En el caso que se desee colocar una piedra en una posición inválida el programa notificará al jugador de que está realizando un movimiento inválido y se anulará la colocación de la piedra, manteniendo el turno del jugador al que le correspondía jugar.

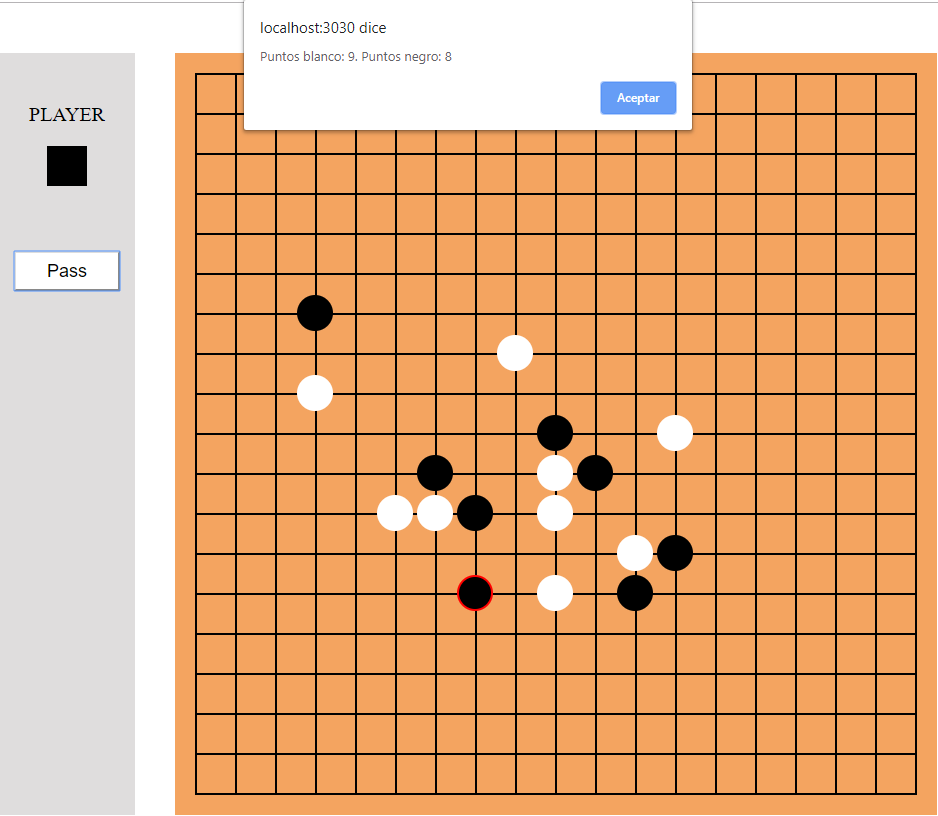
## Interfaz del programa



## Colocación de una piedra en una posición invalida



## Finalización de partida y cálculo de puntajes



# Desarrollo del Programa

Se optó la convención de describir cada regla o predicado presentados en el programa, con el objetivo de realizar una descripción clara y concisa de cada uno.

**Comunicación entre interfaz web y programa en Prolog.**

Sea una regla o predicado a modo de ejemplo: predicado(?X, +Y, -Z)

El parámetro ?X representa que X será utilizado…

El parámetro +Y representa que Y será utilizado como dato de entrada y será utilizado dentro de la regla o predicado.

El parámetro –Z representa que Z será utilizado como dato de salida y contendrá un valor de interés una vez finalizada la regla o el predicado.

**Definición de algunos valores utilizados en el programa**:

* “o” : se utilizó para representar una posición que se encuentre fuera del tablero de juego. El objetivo es representar los vecinos de las piedras que se encuentren colocadas en los bordes de dicho tablero.
* “b” : se utilizó para representar al jugador que utiliza las piedras de color negro.
* “w” : se utilizó para representar al jugador que utiliza las piedras de color blanco.

Cabe destacar que cuando se mencionan los términos *posición* o *index*, se hace referencia a un lugar del tablero de la forma [FILA,COLUMNA].

## checked(Player, Index)

Predicado definido de forma dinámica. Se utiliza para marcar como chequeada una posición visitada del tablero de juego mientras el jugador, el parámetro Player, está buscando qué posiciones ha encerrado ya. En este caso la posición visitada será Index.

## encerradoActual(Player, Index)

Predicado definido de forma dinámica. Se utiliza para marcar que una posición del tablero ha sido encerrada por el jugador, en este caso el parámetro Player. La posición a encerrar en si es el parámetro Index.

## noEncerrado(Index)

Predicado definido de forma dinámica. Se utiliza para marcar que una posición del tablero efectivamente no está encerrada, dicha posición es el parámetro Index. El objetivo de este predicado es facilitar y efectivizar el conteo de puntos, ya que se utiliza una vez haya finalizado la partida y se tiene la certeza de que el tablero no volverá a cambiar.

## emptyBoard(Board)

Predicado destinado a generar un tablero vacío. Genera un tablero vacío en el parámetro Board.

## goMove(+Board, +Player, +Pos, -RBoard)

Predicado destinado a reflejar lo causado por el movimiento de un jugador. Se reciben como parámetros el tablero en su estado actual, en este caso Board, el jugador a realizar el movimiento, recibido como Player, y la posición en la que Player desea colocar una piedra, la cual se recibe como Pos. Una vez colocada la piedra de Player en la posición, Pos, que este haya elegido, se procede a analizar si con este movimiento Player encerró a alguna piedra del jugador contrario. En caso de ser así, se procede a eliminar la/s piedra/s y generar un nuevo tablero con estos cambios reflejados, dicho tablero es RBoard, en el caso contrario se devuelve en RBoard un nuevo tablero con la jugada realizada. Luego se verifica que la posición en la que se deseó colocar la piedra no haya sido un suicidio, en caso de ser así se devuelve en RBoard un nuevo tablero con los cambios reflejados, caso contrario el predicado devuelve false.

## replace(?X, +XIndex, +Y, +Xs, -XsY)

Predicado utilizado para reemplazar en la lista Xs, en el índice XIndex, al elemento X por el elemento Y, devolviendo el resultado en la lista XsY. Dicho predicado fue implementado por la cátedra.

## esValida(+Board, +Player, +Index)

Predicado destinado a analizar si la posición (Index) en la que el jugador (Player) desea colocar una piedra es válida, es decir no está cometiendo suicidio. Se utiliza el tablero (Board) para recorrerlo e ir chequeando las posiciones recorridas. Una vez finalizado el predicado se proceden a deschequear todas las posiciones marcadas.

## buscarEliminarVecinosEncerrrados(+Board, +Index, +Player, -NBoard)

Predicado utilizado para analizar si los vecinos, es decir las posiciones adyacentes, de una dada posición(Index) están encerrados, de ser así se los marcará como chequeados, de la forma encerrado. Se utilizará el tablero (Board) para analizar y recorrer todas las piedras, y se utilizará al jugador (Player) para decider si el jugador que está encerrando es el de piedras blancas o negras. Una vez finalizado el predicado se procederá a eliminar todas las piedras marcadas como encerradas y se retornará un nuevo tablero con los cambios reflejados, el cual es el parámetro NBoard.

## checkEncerrado(+Board, +Index, +Player, +Opponent, +Liberty, -NBoard)

Predicado destinado a analizar si los vecinos de una posición (Index) se encuentran encerrados. Se utilizará el tablero actual (Board) para recorrer y analizar las posiciones, como también al jugador (Player) y el jugador que juega encontra (Opponent), además se utilizará un valor de libertad para analizar si la posición se encuentra libre, es decir no está ocupada por ningun tipo de piedra, este valor se pasa como parámetro con el nombre de Liberty. Si analizando los vecinos se encuentra con un valor de Liberty se procede a finalizar el recorrido ya que una vez encontrado esto se concluye que no se encuentra encerrado el Index. En el caso de eliminar piedra/s se generará un nuevo tablero con los cambios efectuados con el nombre de NBoard, caso contrario se devolvera el tablero original (Board) dentro del parametro NBoard.

## checkEncerradoCascara(+Board, +Player, +Opponent, +Liberty, -NBoard, +Index)

## estaEncerrado(+Board, +Player, +Opponent, +Liberty, +Index)

## getVecinos(+Index, -IndexesVecinos)

Predicado utilizado para obtener los vecinos, es decir posiciones adyacentes, de una posición dada (Index). Dichos vecinos se retornarán en una lista llamada IndexesVecinos.

## vecinoEstaEncerrado(+Board, +Player, +Opponent, +Liberty, +ValuesVecinos, +IndexVecinos)

## noEstaEncerrado(+Board, +Player, +Opponent, +Index)

## getValueListOnBoard(+Board, +Indexes, -Values)

## getValueOnBoard(+Board, +Index, -Value)

Predicado destinado a la obtención del valor de una posición dada (Index). Se utiliza el tablero actual (Board) para acceder a dicha posición. El valor obtenido se retornará en el parámetro Value.

## getValueLista(+Columna, +Lista, -Value)

Predicado utilizado para la obtención del valor de la posición de la columna pasada por parametro ubicada en la lista (Lista). En el caso de que se haya encontrado una posición valida, se procede a retornar su valor en el parámetro Value, caso contrario se procede a devolver el valor “o” en Value.

## getListaIndex(+Index, +Board, -Lista)

Predicado destinado a la obtención de la lista correspondiente a la fila representada por el parámetro Index. Se utilizará el tablero actual (Board) para recorrer la fila y obtener sus respectivos elementos. En el caso de ingresar un Index valido, se procederá a retornar la correspondiente lista en el parámetro Lista, caso contrario se retornará en el parámetro Lista una lista vacía.

## eliminarEncerradosActuales(+Board, -NBoard)

Predicado utilizado para eliminar todas las posiciones marcadas como encerrados actuales del tablero actual (Board), luego se retorna un nuevo tablero que refleje todos los cambios aplicados en NBoard.

## winPoints(+Board, -PWhite, -PBlack)

Predicado destinado a calcular el puntaje de ambos jugadores una vez finalizada la partida. El puntaje de cada jugador se calcula contando las posiciones en el tablero ocupadas por cada uno, sumado a la cantidad de posiciones que se encuentren vacías y que esten capturadas por un jugador. Se recibe el tablero actual (Board) como parámetro para poder recorrerlo y calcular dichos puntajes. El puntaje de las piedras de color blanco se retorna en el parámetro PWhite y el puntaje de las piedras negras se retorna en el parámetro PBlack.

## calcularPuntos(+Board, +Player, +Opponent, +Liberty, -P)

## calcularPuntosAux(Board, Player, Opponent, Liberty, Index)

## checkEncerradoSinUncheck(+Board, +Player, +Opponent, +Liberty, +Index)

## getNext(+Index, -NextIndex)

Predicado utilizado para obtener el index o posición siguiente a un Index pasado como parámetro. En el caso de que Index no se encuentre en el extremo derecho de su fila, se retornará en NextIndex la posición a la derecha de Index, caso contrario se retornará la primera posición de la fila siguiente en NextIndex.

## checkedToEncerrado

## checkedToNoEncerrado

# Conclusión

El desarrollo de este proyecto nos ha permitido incorporar nuevos conceptos sobre el ámbito lógico y cómo este se relaciona con el campo de las ciencias de la computación, además nos introdujo a un nuevo paradigma de programación, el paradigma lógico. También se aprendió de una manera general, el funcionamiento entre un servidor, la internet y un cliente, y los lenguajes involucrados con estos procedimientos.

**Decidir cual de los dos párrafos queda mejor**

El proyecto y toda su implementación nos brindó una gran variedad de métodos de resolución y pensamiento de problemas, lo cual creemos son una herramienta muy valiosa y útil que nos ayuda a crecer y mejorar como profesionales.

El proyecto y sus requisitos conllevó una valiosa experiencia de aprendizaje para nosotros, ya que nos obligó a pensar cómo resolver problemas con una orientación a la que no estábamos acostumbrados, y creemos que esto nos ayudará a convertirnos en mejores profesionales a futuro.