Universidad de Buenos Aires
Facultad de Ingeniería

75.52 Taller de Programación II
1^{er} Cuatrimestre 2011

Trabajo práctico Go

Fecha de entrega: 3 de Junio

Tutora: Patricia Calvo

Integrantes:

Apellido,Nombre	Padrón Nro.	E-mail
Bukaczewski, Veronica	86.954	vero13@gmail.com
De Antoni, Matías	88.506	mdeantoni87@gmail.com
Garbarini, Lucía	88.300	lu.teddy@gmail.com
Pandolfo, Lucas	88.581	lucashpandolfo@gmail.com

Índice

1. Objetivo

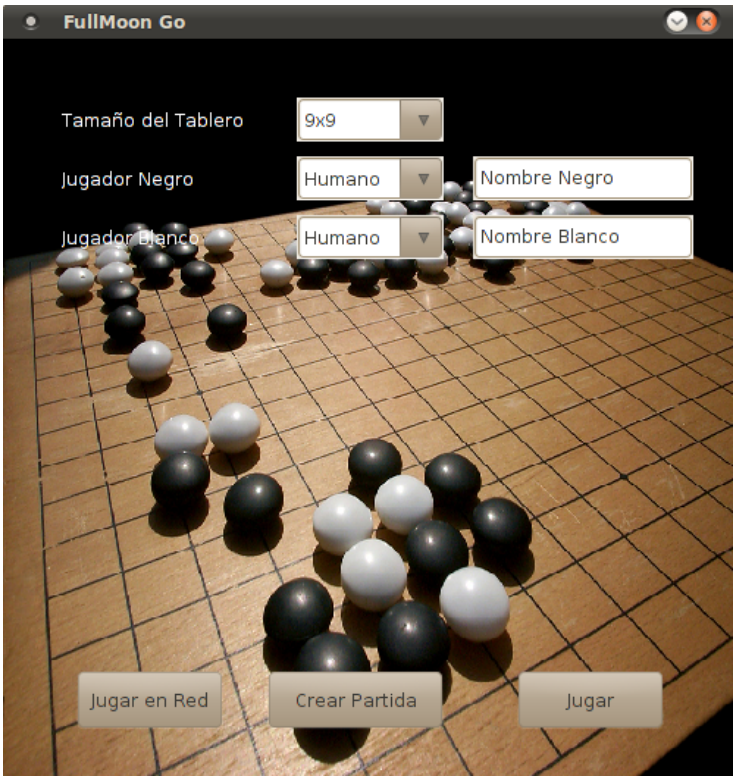
Desarrollar un sistema que permita jugar partidas de **Go**, en un tablero reducido y considerando el juego finalizado **a la primera muerte** (capture Go).

2. Requerimientos funcionales

El sistema debe permitir a dos jugadores humanos en la misma computadora jugar entre si. También debe permitir como posibles participantes del juego a alguna aplicación externa (como ser **gnugo**). Adicionalmente se deben incluir estrategias de juego para que una sola persona pueda desarrollar una partida contra el sistema.

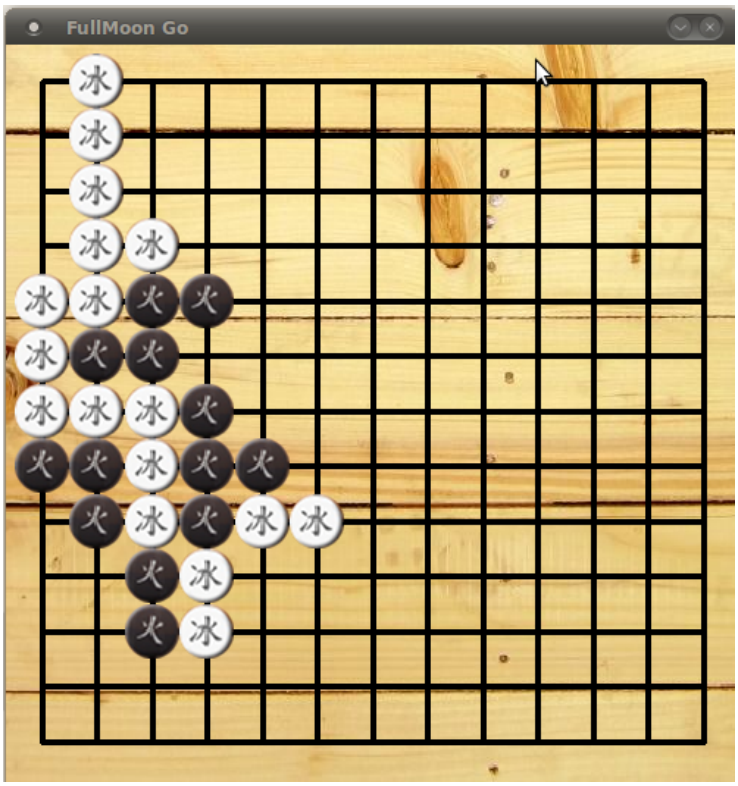
3. Manual de usuario

Menú de inicio



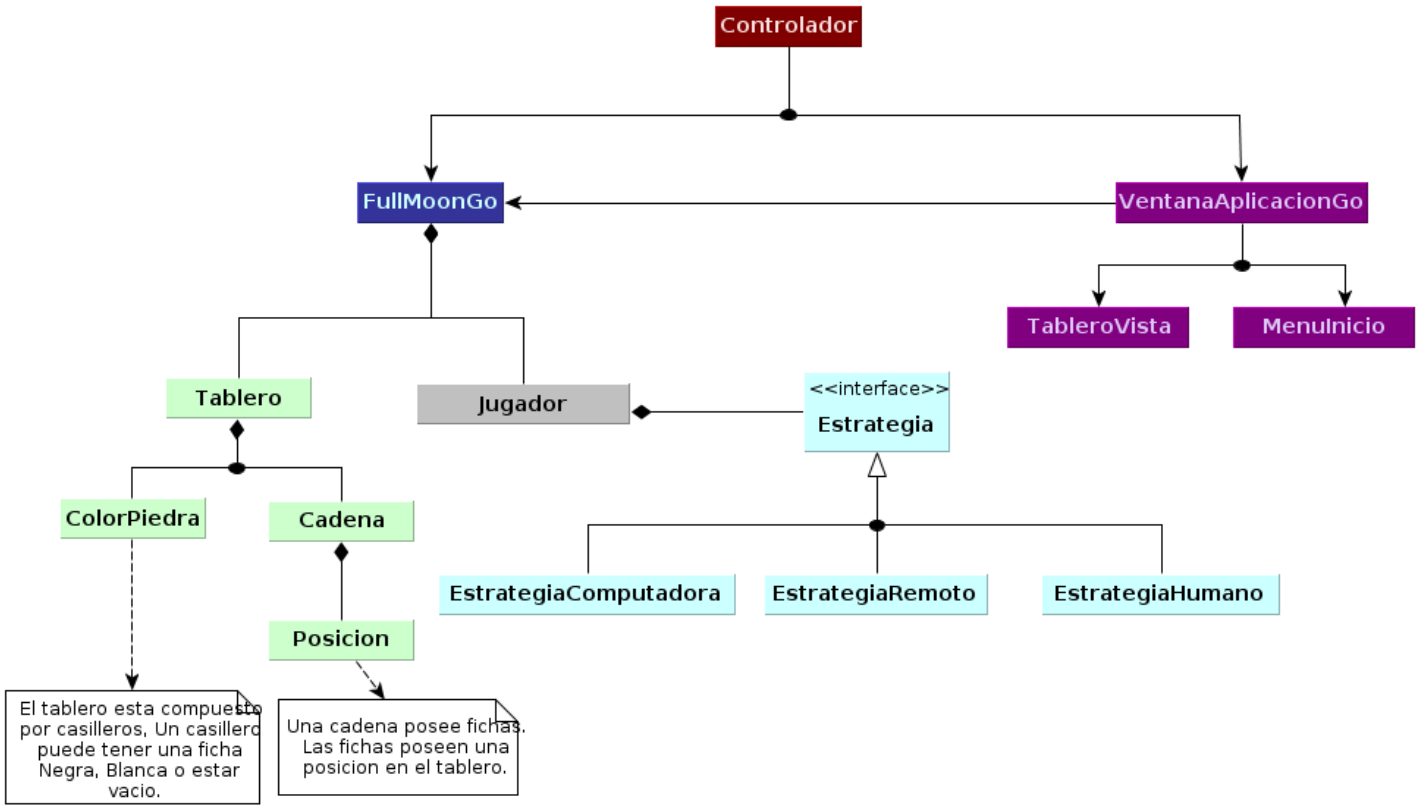
En el menú de inicio se modifica el tamaño del tablero para jugar, las estrategias de cada jugador y el nombre de cada jugador. Para jugar una partida local, luego de seleccionar el tablero y los jugadores, se debe presionar el botón “Jugar”. Si la estrategia elegida es “Humano”, las piedras deben ubicarse en el tablero haciendo click con el botón izquierdo del mouse.
Para jugar una partida en red, uno de los jugadores debe crear un servidor mediante el botón “Crear Partida” donde se pide el puerto por el cual esperar una conexión. El cliente luego puede unirse mediante el botón “Jugar en Red” especificando la ip y el puerto del servidor.
Al jugar en red, el cliente juega con el color negro y el servidor con el color blanco, y es el cliente quien decide el tamaño del tablero con el cual jugar.

Tablero del Juego



4. Detalles de implementación

Estructura general de la solución:



La clase FullMoonGo posee la lógica de control del juego, seleccionando al jugador correspondiente según el turno y detectando el fin de la partida. El tablero es quien verifica la validez de una jugada, encapsulando toda la lógica de cadenas y grados de libertad de una piedra. Los jugadores ubica una piedra en el tablero, eligiendo su jugada en base a su estrategia asociada, la cual puede ser:

- EstrategiaHumano
  Representa la jugada realizada por un humano mediante acciones con el mouse sobre la vista.
- EstrategiaRemota
  Obtiene la jugada realizada por un jugador conectado en red.
- EstrategiaComputadora
  Aplica un algoritmo para decidir en qué posición del tablero jugar.

Por otro lado, el controlador interactúa con la vista, indicándole al modelo qué estrategia asignar a cada jugador y cuándo iniciar la partida. En cuanto a una partida remota, el controlador le indica al árbitro de la partida (quien maneja la interacción con el usuario remoto) cuando crear una conexión, ya sea como cliente o como servidor.

4.1. Vista

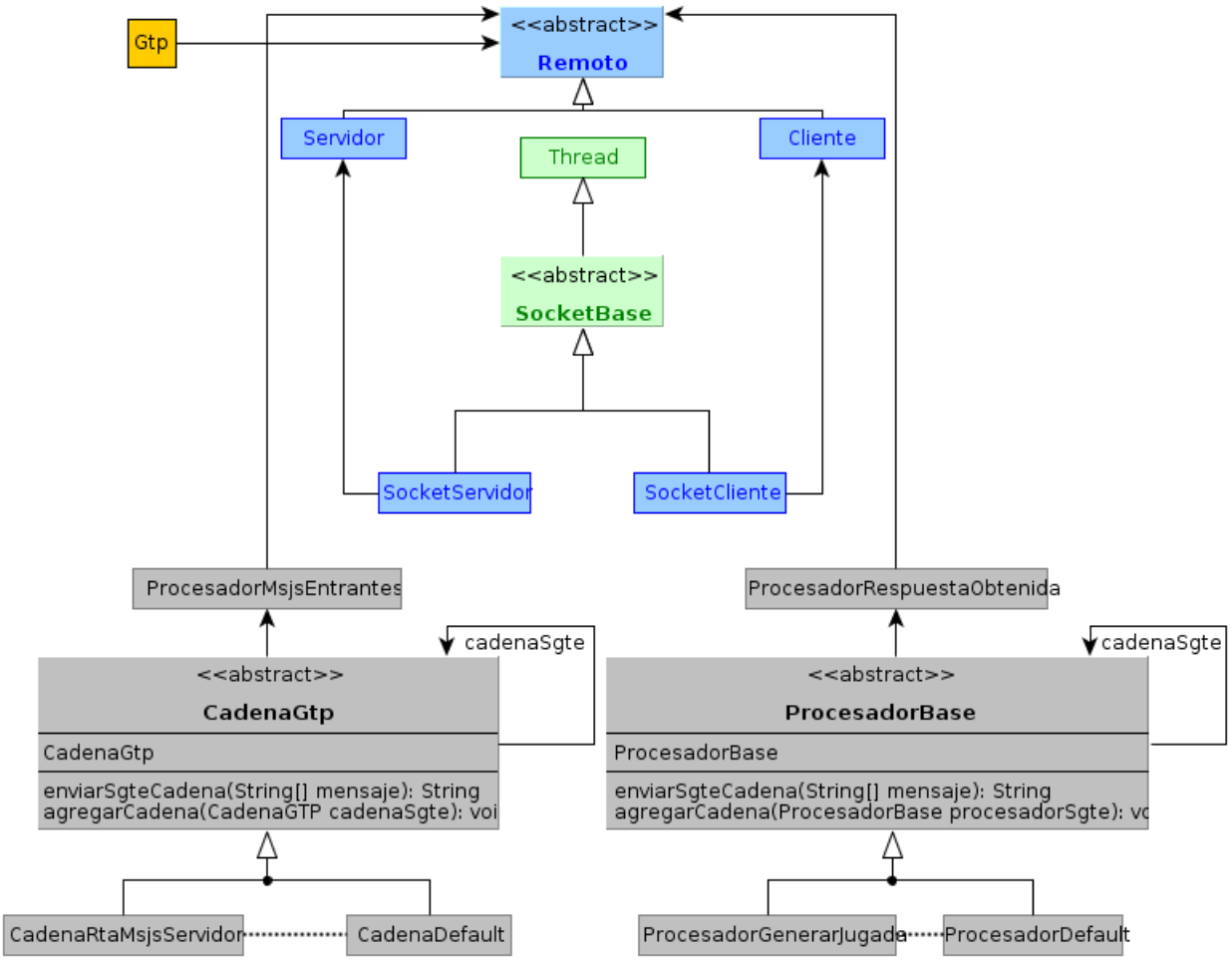
Para implementar el aspecto visual de la aplicación se uso el la api de Java Swing. Principalmente se utilizaron las clases Frame como contenedor principal y Panel para las distintas

pantallas. Estas son menú de inicio y la vista del tablero. En el menú se permite elegir la estrategia de cada jugador y se da la opción de crear una partida como servidor o unirse a una partida remota como cliente. En cuanto a la vista del tablero, ésta muestra el estado actual del tablero.
La vista no posee lógica alguna, sino que detrás de cada una hay un controlador que toma la información y la utiliza según corresponda. De la misma forma, es el controlador quien espera cambios en el modelo para luego indicarle a la vista que debe actualizarse. Para implementar este mecanismo de comunicación de se utilizó el patrón Observer provisto por la API de Java. Los subscritores son las clases de Control y las clases Observadas son las correspondientes al modelo.

Otro aspecto adicional de las clases de la vista es que manejan la reproducción de los sonidos de posicionamiento de piezas. Siendo uno distinto para cada pieza. Para esto se uso la clase clip que resulta adecuada para una reproducción repetida de sonidos cortos pues los mantiene cargados (En contraposicion a los Flujos de datos.

4.2. Protocolo de comunicaciones

La clase remoto es la encargada del manejo del protocolo de texto Go(gtp), si el modelo desea enviar un mensaje lo debe hacer por esta clase. Como se puede observar en el diagrama servidor y cliente son hijos de remoto; esto se debe a que el proceso de mensajes entrantes y saliente es el mismo en ambos casos, la única diferencia existente es el momento inicial donde se establece la conexión. Para el proceso de los mensajes, tenemos dos cadenas de responsabilidades; una encargada de los mensajes respuestas (los cuales se caracterizan por el inicio con el caracter **=**) y los mensajes comandos. Estas cadenas procesan dichos mensajes y le informan al modelo, para que el mismo decida como se debe continuar.
La versión del protocolo utilizada es la 2, a pesar que el protocolo fue implementado totalmente; para los alcances del trabajo práctico solo se utilizan los comandos más importantes.



4.2.1. EstrategiaRemoto

Para comunicar dos jugadores en red mediante el protocolo, se tiene una clase "EstrategiaRemota" la cual abstrae al motor del juego del intercambio de mensajes entre jugadores.
Para notificar y pedir jugadas mediante el protocolo, se cuenta con una implementación que es compatible con el protocolo GTP para el GnuGo, el cual es un servidor pasivo que simplemente responde a los comandos enviados por GTP. Por esta razón, implementamos separadamente la estrategia remota para el servidor y para el cliente, para poder adaptarlos a los mensajes que deben enviar y los que esperan recibir.
La convención de mensajes utilizada es la siguiente:

- EstrategiaRemotaCliente
  Se conecta con un servidor y determina el tamaño del tablero mediante un mensaje *"boardsize"*.
  En el turno del jugador remoto, se envía la jugada realizada localmente mediante el mensaje *"play"*, luego se solicita la jugada al servidor remoto mediante un mensaje *"genmove"* y se aguarda hasta recibir una respuesta. Cuando esta respuesta llega, el jugador remoto puede poner la piedra recibida en el tablero.

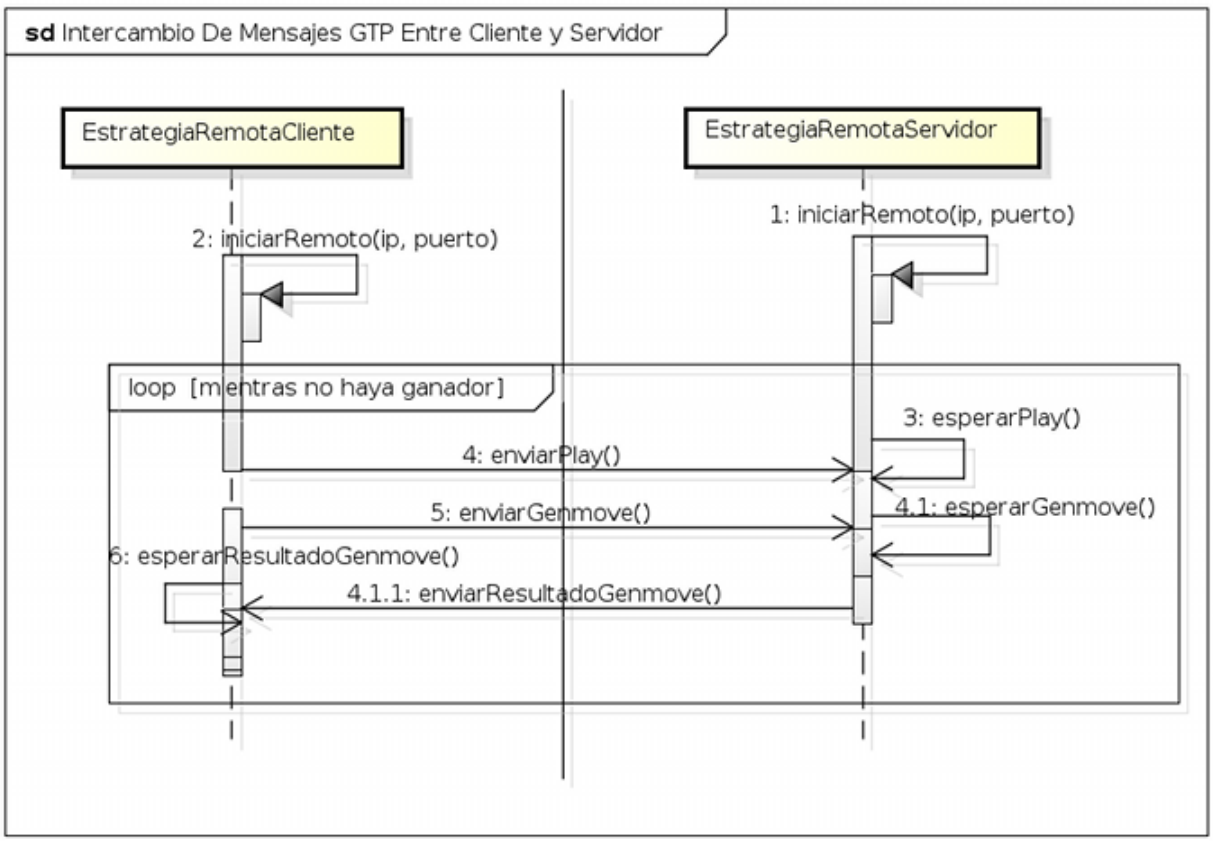
- EstrategiaRemotaServidor

Crea un servidor remoto que espera la conexión de un cliente. El primer mensaje debe ser un *boardize* para determinar el tablero del juego. Dado que el servidor es pasivo, cuando es el turno del jugador remoto la estrategia espera la llegada de un mensaje *"genmove"*, el cual contesta con la última jugada local realizada. Luego, el servidor queda a la espera de la llegada de un mensaje *"play"* el cual indica la jugada remota a realizar.

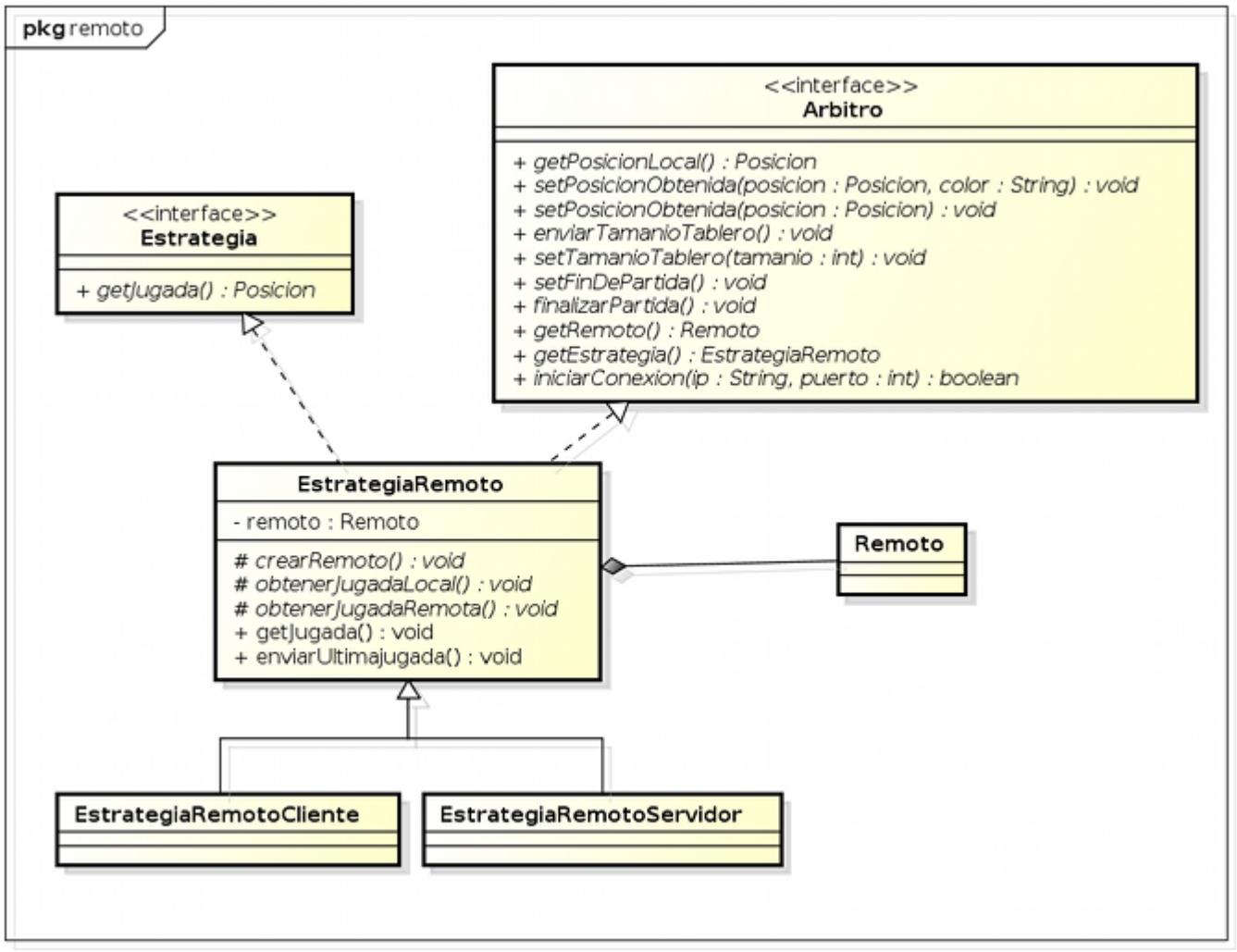
- Fin Del Juego
  Al finalizar el juego (alguno de los dos jugadores gana), es el cliente quien envía el mensaje de *"quit"* para finalizar la comunicación mediante GTP.

- Desconexión
  Ante una desconexión repentina de alguno de los dos extremos, la partida finaliza mostrando un mensaje de empate.

A continuación se incluye un diagrama de secuencia describiendo la interacción mencionada.



La estrategia remota realiza el intercambio de mensajes utilizando la implementación de la clase "Remoto" que corresponda (cliente o servidor). Además, funciona como árbitro de la partida mediante GTP, coordinando los mensajes administrativos como son el mensaje de salida (*quit*) y el mensaje indicando el tablero (*boardisze*).


pkg remoto
<<interface>>
Arbitro
+ getPosicionLocal() : Posicion
+ setPosicionObtenida(posicion : Posicion, color : String) : void
+ setPosicionObtenida(posicion : Posicion) : void
+ enviarTamanioTablero() : void
+ setTamanioTablero(tamanio : int) : void
+ setFinDePartida() : void
+ finalizarPartida() : void
+ getRemoto() : Remoto
+ getEstrategia() : EstrategiaRemoto
+ iniciarConexion(ip : String, puerto : int) : boolean
<<interface>>
Estrategia
+ getJugada() : Posicion
EstrategiaRemoto
- remoto : Remoto
crearRemoto() : void
obtenerJugadaLocal() : void
obtenerJugadaRemota() : void
+ getJugada() : void
+ enviarUltimajugada() : void
Remoto
EstrategiaRemotoCliente
EstrategiaRemotoServidor

4.3. Estrategias implementadas

Actualmente se cuenta con cuatro estrategias de juego que serán utilizadas posteriormente para elaborar estrategias mas avanzadas:

4.3.1. EstrategiaComputadoraAtacar

Esta estrategia intenta ocupar casilleros adyacentes a las cadenas con menor grado de libertad del oponente, intentando capturarlas.

4.3.2. EstrategiaComputadoraDefender

Esta estrategia intenta ocupar casilleros adyacentes a las cadenas propias con menor grado de libertad, intentando evitar que sean capturadas.

4.3.3. EstrategiaAtaqueCuidadoso

Esta estrategia es una combinación de las estrategias "EstrategiaComputadoraAtacar" y "EstrategiaComputadoraDefender". Primero verifica que las cadenas propias no estén en peligro de ser capturadas. Si se encuentra una cadena propia en riesgo aplica la estrategia "EstrategiaComputadoraDefender", en caso contrario aplica "EstrategiaComputadoraAtacar".

4.3.4. EstrategiaAtaqueCuidadosoMasInteligente

Similar a la estrategia anterior, pero primero verifica si existe alguna cadena del oponente con grado de libertad 1. Si existe, verifica que ese grado de libertad no se deba a un ojo. Si no se deba a un ojo se procede a capturar al grupo. Si no se cumplen estas condiciones, se aplica la estrategia "EstrategiaAtaqueCuidadoso".

4.3.5. EstrategiaMiniMax

Implementa una estrategia del tipo **MiniMax**. En cada turno, arma una lista de todos los casilleros vacíos y despliega un árbol de jugadas por cada posible casillero. La profundidad hasta la cual despliega elárbol de jugadas es configurable. Al llegar a la profundidad deseada, se aplica la función de evaluación a los tableros resultantes.

La función de evaluación tiene en cuenta las siguientes variables:

- Grados de libertad de MAX: Se cuentan todos los casilleros adyacentes a cada cadena de MAX libres (no se cuentan los repetidos). Se quiere que sea lo mayor posible.
- Cantidad de ojos de MAX
- Grados de libertad de la cadena mas corta de MAX: la variable a la que se le da más importancia.
- Grados de libertad de la cadena mas corta de MIN.
- Grados de libertad de la cadena mas larga de MIN.
- Ojos de MIN.

Por encima de las variables arriba mencionadas, se encuentra la condición de que alguna de alguna de las cadenas de MIN tenga grado 1. En ese caso se da por ganada la partida (en esa rama del árbol de jugadas).

Al desplegar el árbol de jugadas, si en algún nivel todos los movimientos son inválidos, se da por finalizada la partida y no se sigue avanzando hasta los niveles mas profundos.

4.3.6. Mejoras posibles

La implementación MiniMax presentada se puede mejorar teniendo en cuenta lo siguiente:

- Si se es el primero en jugar, no es necesario descender en el árbol de jugadas. Sería conveniente comenzar con jugadas precalculadas.
- Se tienen en cuenta como jugadas posibles todos los casilleros libres del tablero. Se podría tener en cuenta solamente los casilleros adyacentes a todas las cadenas presentes en el tablero solamente, limitando así las jugadas, pero reduciendo el procesamiento, eventualmente dándonos la posibilidad de descender un poco mas en el árbol de jugadas.
- La cantidad de niveles que se baja en el árbol de jugadas es fija. Se puede parametrizar en función de las jugadas posibles. Al principio de la partida, al haber muchas posibilidades se elige por ejemplo descender hasta el nivel 3, pero a medida que quedan menos posibilidades (la mitad del tablero puede ser un caso) podríamos aumentar un nivel sin perder mucha velocidad de respuesta.
- La función de evaluación actualmente es una suma pesada de diferentes variables. Se pueden implementar diferentes funciones de evaluación mas sofisticadas.