Universidad de Buenos Aires
Facultad de Ingeniería

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

Trabajo práctico Go

Fecha de entrega: 24 de Junio 2011

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

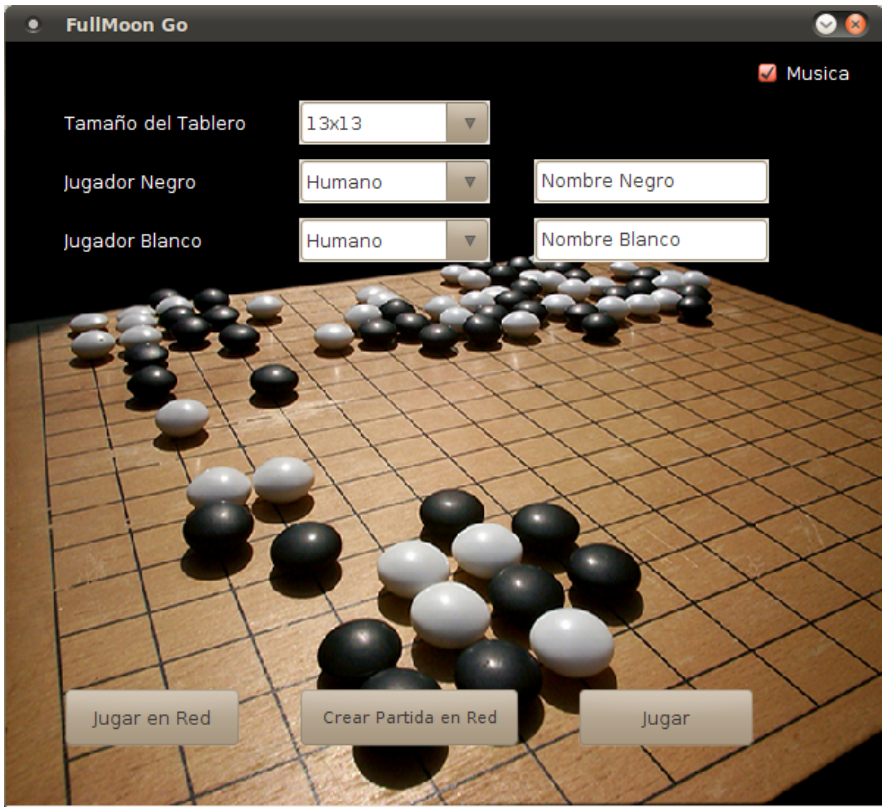
Desarrollar un sistema que permita jugar partidas de **Go**, en un tablero reducido y considerando el juego finalizado **a la primera muerte** (capture Go).

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

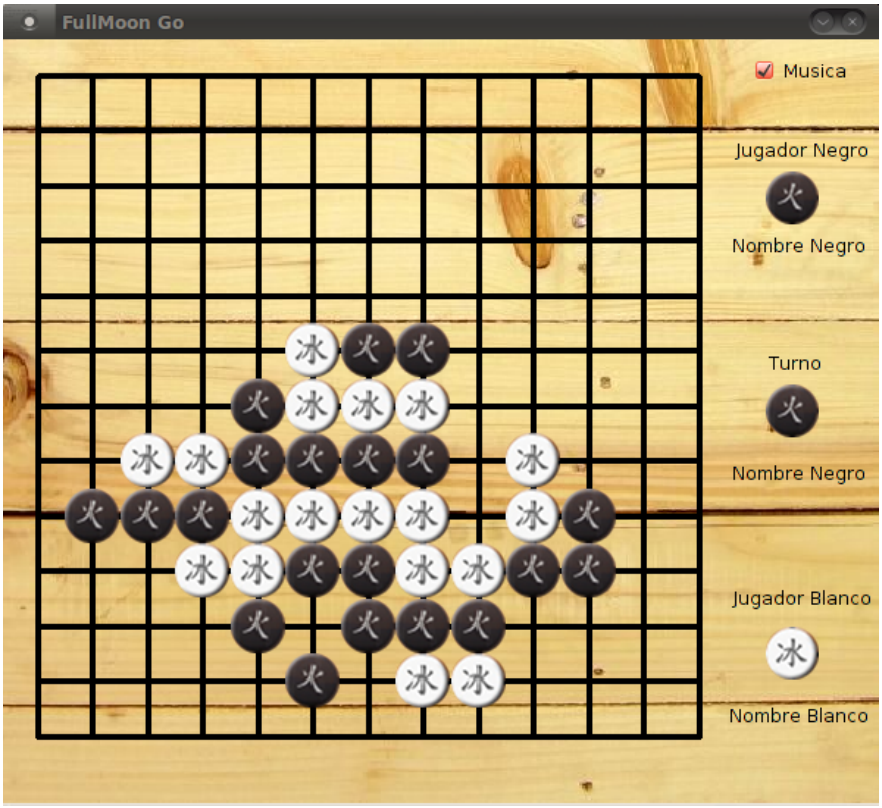


En el menú de inicio se modifica el tamaño del tablero para jugar, las estrategias de cada jugador y el nombre de cada jugador. Para jugar una partida local, luego de seleccionar el tablero y los jugadores, se debe presionar el botón “Jugar”. Las estrategias pueden ser “Humano”, que es el usuario el que debe realizar las jugadas, o bien alguna de las estrategias desarrolladas.
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. La estrategia a utilizar en la partida remota será la elegida en el color de jugador correspondiente.

Se tiene también un checkbox para habilitar o deshabilitar el sonido, tanto en el menú como en el tablero del juego.

Tablero del Juego



Para jugar se deben posicionar las piedras mediante el click izquierdo del mouse (si la estrategia elegida es “Humano”). Para *pasar el turno* se debe presionar el *boón derecho* del mouse. Las reglas del Go indican que si los dos jugadores pasan consecutivamente el juego termina.
A la derecha del tablero se pueden ver los datos de los jugadores de la partida. Además se indica el jugador de turno que debe realizar la proxima jugada.

4. Evolución del Proyecto

A continuación se detalla la evolución del proyecto a lo largo del cuatrimestre.

- **11 de Marzo** Presentación en clase del Proyecto a Realizar
- **Semana 1 y 2** - Análisis del Go.
  Dado que la mayoría de nosotros no conocíamos el juego, durante las primeras semanas nos dedicamos aprender a jugar para comprender la mecánica del juego y desarrollar estrategias eficientes más adelante en el proyecto.
- **Semana 3 y 4** - Reglas de Juego y análisis de la solución
  Durante esta etapa analizamos la problemática planteada y diagramamos la estructura general de la solución.
  Iniciamos con la implementación del tablero, el cual arma cadenas (o grupos) de piedras y valida las reglas del juego.
- **Semana 5 y 6** - Comienzo de interfaz gráfica
  Continuamos con el desarrollo del tablero y la creación de tests unitarios para validar su correcto funcionamiento.
  Comenzamos a desarrollar una interfaz gráfica para visualizar el tablero y poder empezar jugar "player vs player".
- **Semana 7 y 8** - Primeras Estrategias y Protocolo de Comunicación
  Comenzamos desarrollando estrategias simples (algunas de las cuales ahora quedaron como estrategias "fáciles").
  Seguimos con el desarrollo de la interfaz gráfica y empezamos a implementar el procesador de comandos de GTP, que es el protocolo de comunicación utilizado.
- **Semana 9 y 10** - Avance en Estrategias y Protocolo
  Agregamos la funcionalidad cliente/servidor para la implementación del protocolo GTP.
  Empezamos con el desarrollo de la estrategia MiniMax.
- **20 de Mayo** - Presentación de avances
  Mostramos la funcionalidad básica del juego: partida "player vs. player" y "player vs estrategia"
- **Semana 11** - Interfaz gráfica y Avance en estrategia
  Continuamos mejorando la estrategia MiniMax para que esta sea menos predecible y más rápida al realizar una jugada.
  Creamos un menú de inicio para la selección del tamaño del tablero y de la estrategia deseada para cada jugador.
- **Semana 12** - Juego Remoto
  Integramos la funcionalidad cliente/servidor a la aplicación para jugar partidas remotas contra engines de Go compatibles con el GNUgo en modo GTP.
- **Semana 13** - Avance en estrategia
  Desarrollamos una estrategia en base a MiniMax que llamamos "MiniMax reducido", la cual reduce el tiempo de cálculo considerablemente durante las primeras jugadas al evaluar menor cantidad de jugadas posibles.

- **10 de Junio** - Presentación de avances
  Mostramos la aplicación jugando de forma remota y utilizando las estrategias MiniMax y MiniMax reducido.
- **Semana 14 y 15** - Detalles Finales
  Realizamos los ajustes finales, principalmente a la interfaz gráfica para mostrar información del estado de la partida.
- **24 de Junio** - Presentación Final de la Aplicación - Cierre del Proyecto

5. Estrategias implementadas

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

5.1. EstrategiaComputadoraAtacar

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

5.2. EstrategiaComputadoraDefender

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

5.3. 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".

5.4. 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".

5.5. 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 el árbol 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.

5.6. MiniMax Reducido

El MiniMax anteriormente explicado tiene en cuenta como jugadas posibles todos los casilleros libres del tablero. A diferencia de éste, el MiniMax reducido solamente tiene en cuenta los casilleros adyacentes a las cadenas presentes en el tablero. Si bien se limitan las posibles jugadas, se reduce el procesamiento logrando así una respuesta más rápida de la estrategia (en particular en los primeros turnos que es cuando más casilleros libres hay).

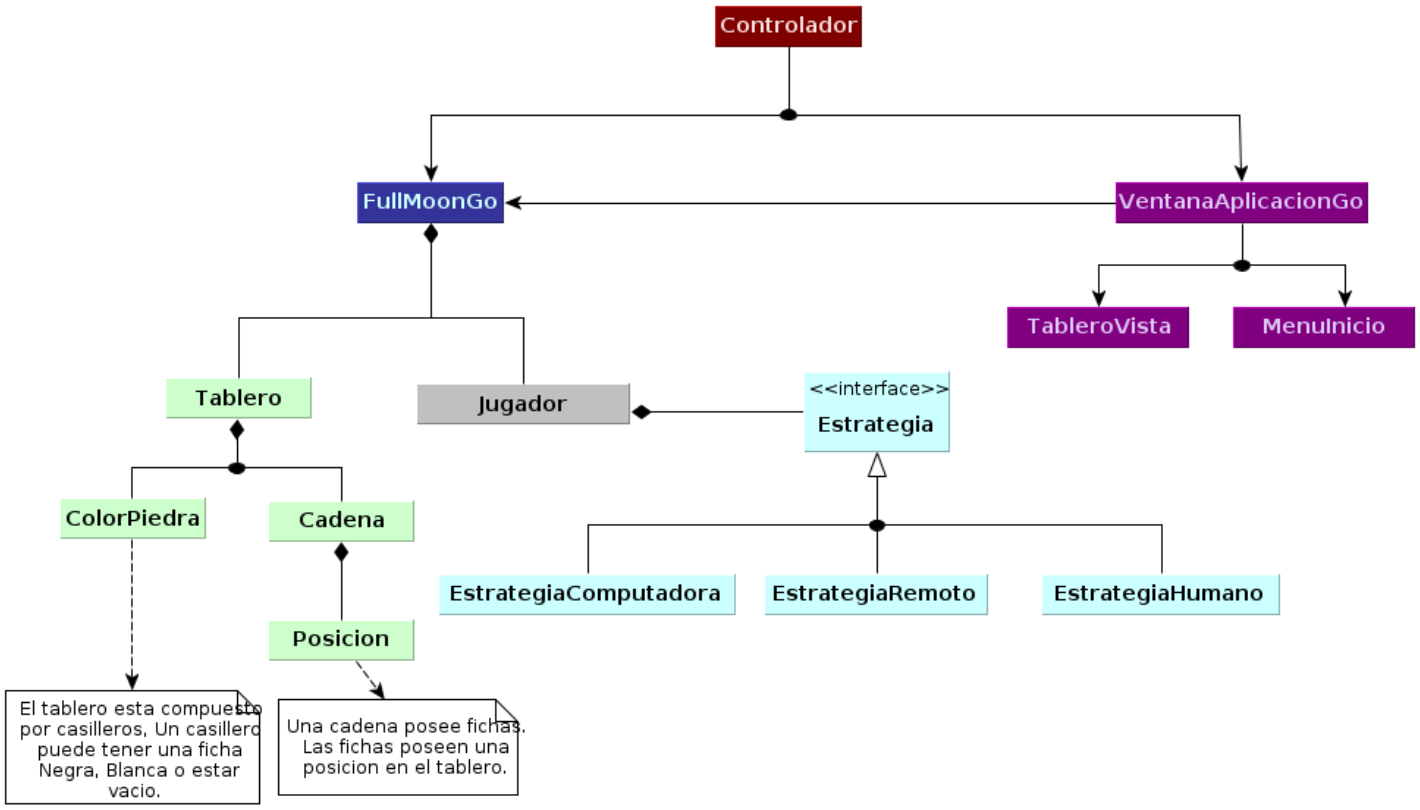
5.6.1. Mejoras posibles

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

- 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.

6. 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.

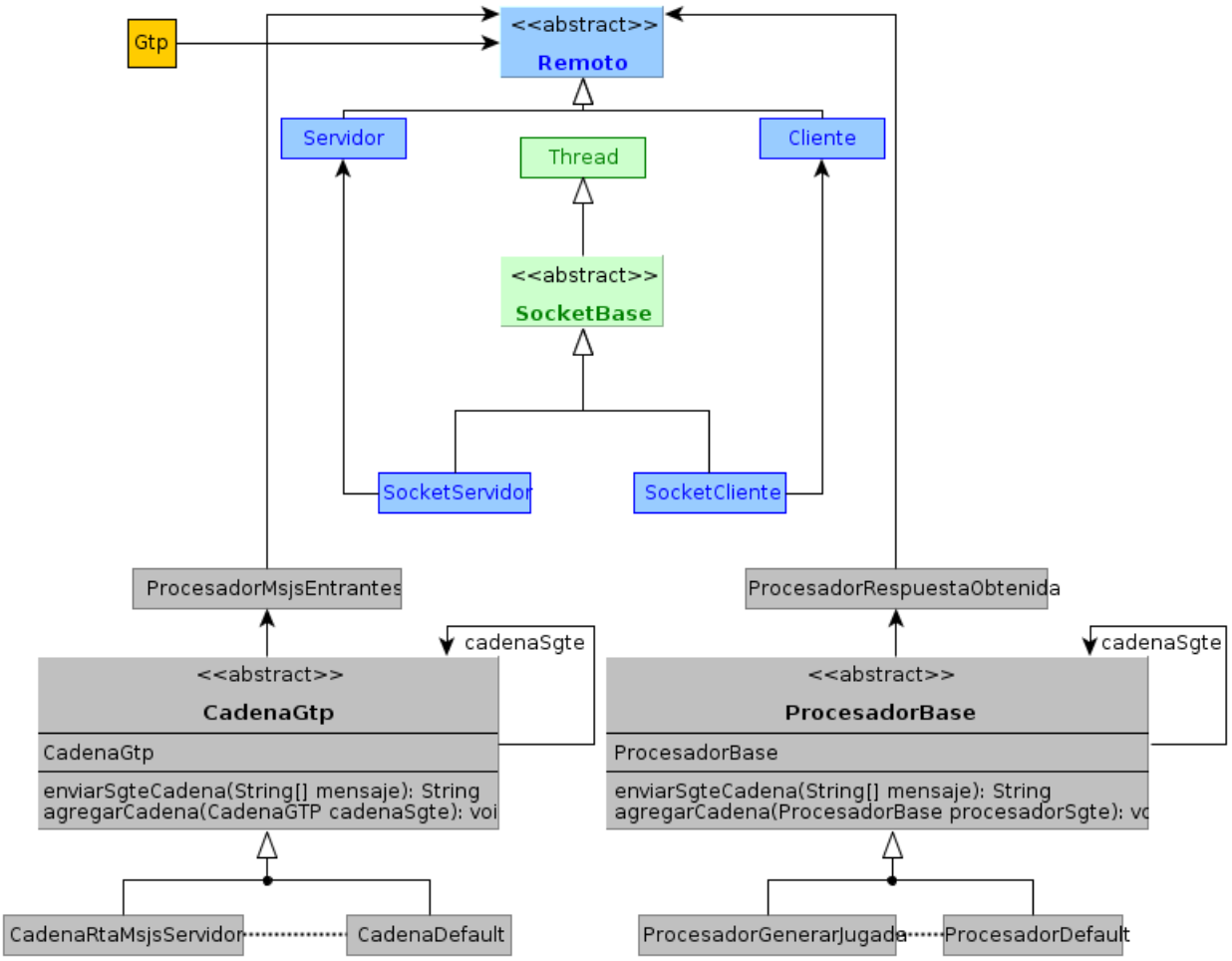
6.1. Vista

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).

6.2. 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.



6.2.1. EstrategiaRemoto

Para comunicar dos jugadores en red mediante el protocolo, se tiene una clase "Estrategia-Remota" 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.

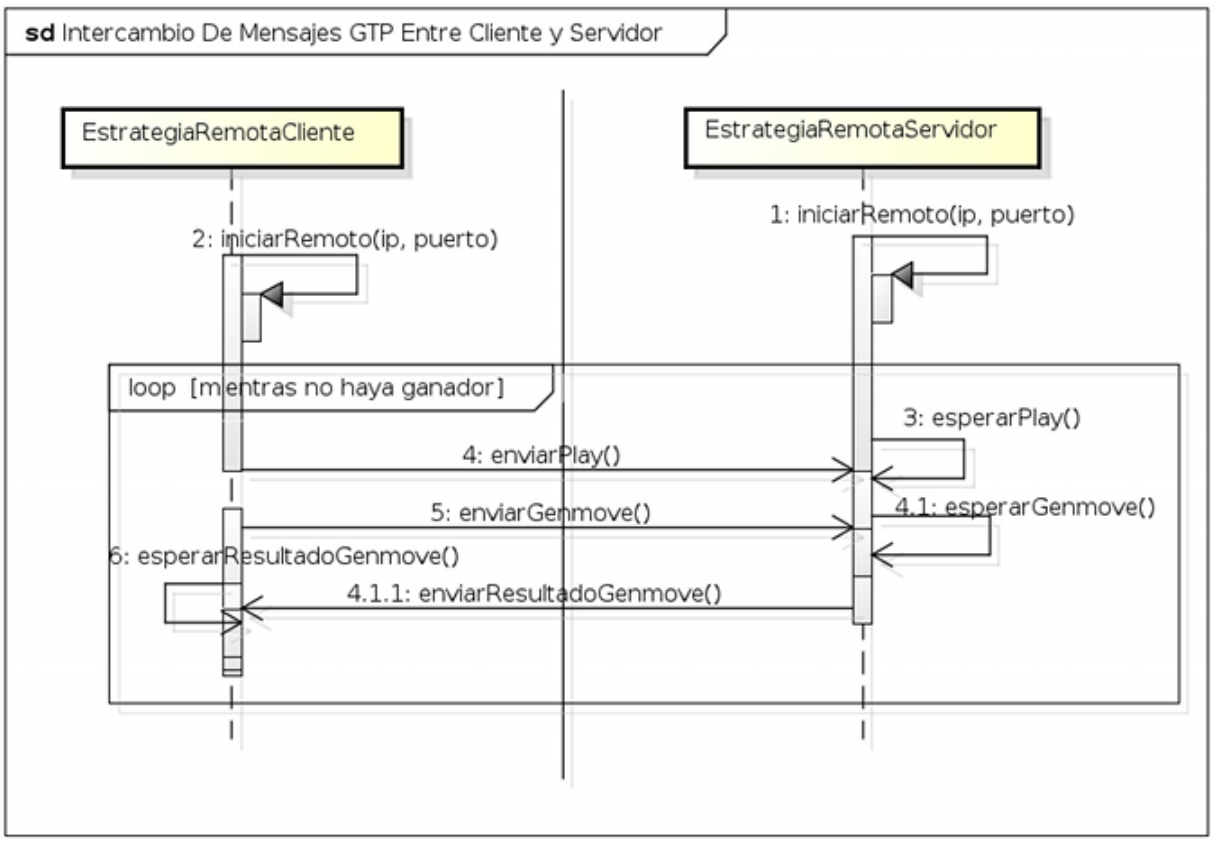
- EstrategiaRemotaServidor

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.

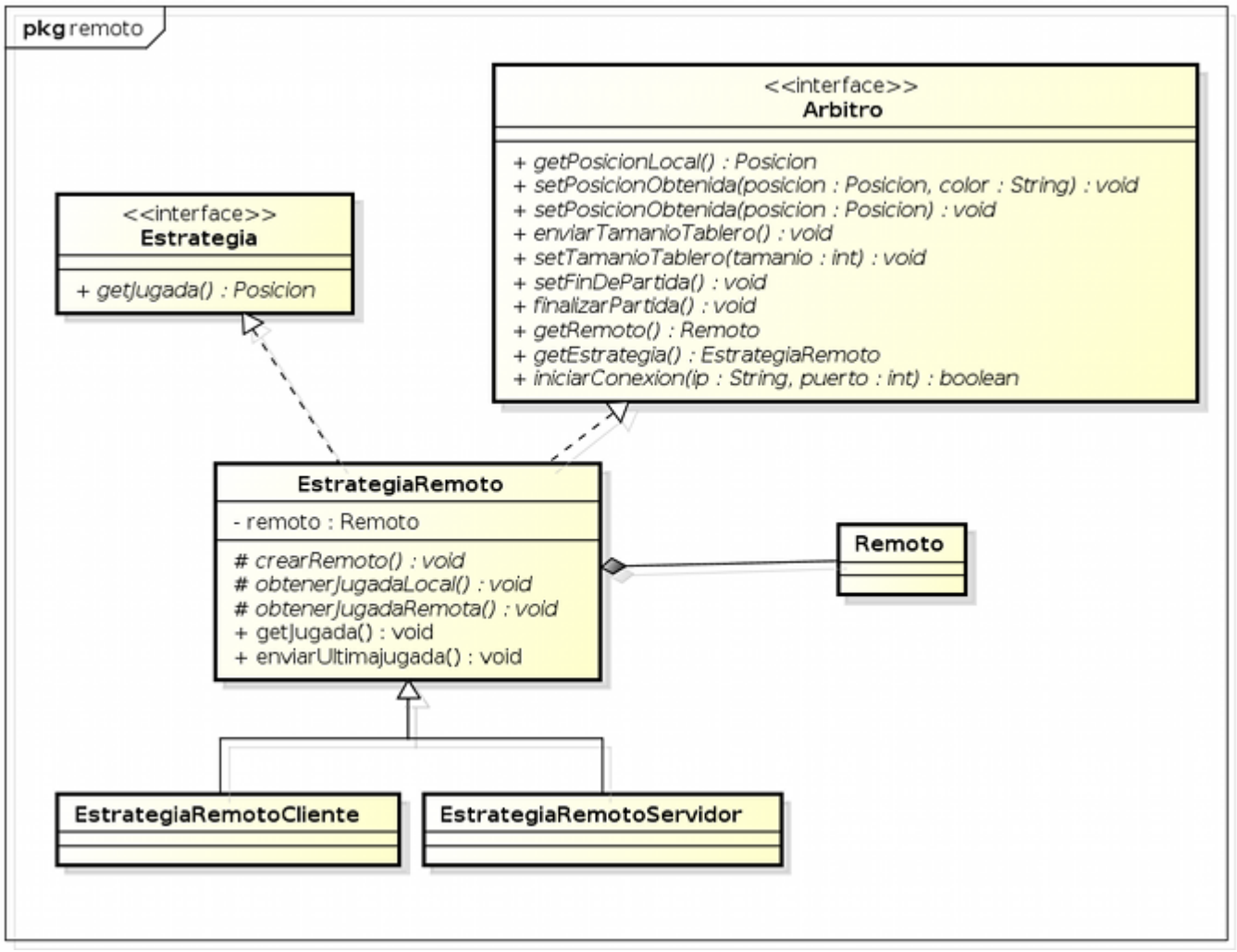
- Fin Del Juego
  Al finalizar el juego (alguno de los dos jugadores gana), es el cliente quien envía el mensaje de *"quit"* para finalizar la comunicación mediante GTP.

- 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

A. Problemas y Soluciones

A lo largo del proyecto nos encontramos con algunos inconvenientes a la hora de desarrollar la aplicación.

- **Protocolo**
  Para implementar la comunicación entre cliente y servidor utilizamos el protocolo GTP, sugerido por la cátedra para compatibilidad entre los distintos engines. Este protocolo presenta distintas variantes al contar con los comandos "gen_move" y "play" para el intercambio de jugadas. El problema surgió al decidir de qué forma se realizaría el intercambio de mensajes y quién validaría las reglas de juego.
  Optamos por una implementación que sea compatible con el GNUgo en modo gtp servidor y así permitir jugar contra un engine distinto del nuestro. Como se explica en la sección 6.2.1, se implementó un servidor "pasivo" que procesa y contesta mensajes (al igual que el GNUgo) y un cliente activo que envía mensajes, ambos integrados con nuestro engine con lo cual siempre se envían jugadas válidas.

- **Sonido**
  Para tener una aplicación más interactiva, decidimos incorporar sonido al ubicar piedras en el tablero. Sin embargo, la carga de sonidos no se realiza correctamente en algunos de los sistemas en los que probamos la aplicación. En algunas computadoras con Ubuntu no se reconocen como válidos los archivos de sonido, mientras que en otras computadoras se reproducen perfectamente. Un aspecto interesante del error encontrado que nos hizo sospechar de un problema en los manejos de los recursos que hace el sistema operativo fue que el error de carga (hablamos de error de *carga* porque se utilizó la clase Clip) se producia con el primer archivo de sonido a cargar con el que se encontraba la aplicación.

- **Estrategias**
  Para resolver el problema de la generación de jugadas decidimos desarrollar estrategias simples como primera instancia, y luego utilizarlas para generar arboles de jugadas mediante MiniMax. Un problema con el que nos encontramos fue el tiempo de procesamiento al profundizar en el árbol de jugadas, que empeora cuanto más grande es el tablero y resulta molesto al momento de jugar.
  Para solucionar este inconveniente, se modificó la estrategia MiniMax tal como se explica en la sección 5.6, la cual presenta una reducción considerable en el tiempo de respuesta, sin bajar de forma notable la "calidad" de la jugada elegida.

- **HashSet**
  Los HashSet, y los Set en general, aseguran que los elementos contenidos sean únicos. Sin embargo, para que funcione correctamente, es necesario no solo redefinir el metodo "equals" del objeto contenido, sino que también es necesario redefinir el método "hashCode". Si no se redefinen ambos, solo se asegura de que no exista dos veces la misa referencia en el Set, independientemente de que dos elementos sean identicos. Al no conocer este detalle del lenguaje nos surgieron varios problemas de performance ya que verificábamos hasta unas 4 veces el mismo casillero al calcular jugadas.