MineSweeperClient

Información general.

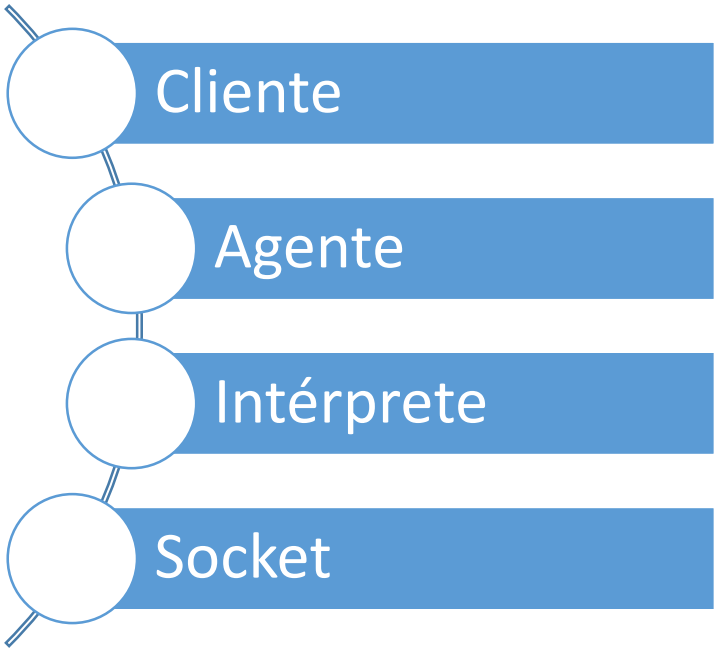
Este sistema es un cliente que permite jugar buscaminas contra otro jugador en el mismo tablero en un servidor externo utilizando sockets UDP.

Objetivo

El objetivo del sistema es simple, **ganar el juego.**

Arquitectura

El sistema utiliza una arquitectura dividida en capas abiertas en la que cada capa sólo puede comunicarse con alguna capa inferior. Esto permite la división de responsabilidades y minimiza la creación de interfaces.



De manera general, el proceso de un ciclo de juego ser realiza cuando el socket recibe una actualización del servidor, éste pasa la información recibida al intérprete que se encarga de convertir el mensaje en objetos que el cliente y el agente usan para tomar decisiones acerca del estado del juego (como actualizar el marcador o devolver una jugada).

El proceso inverso comienza con el cliente determinando que es necesario realizar una jugada, pasándole el control al agente que determina la acción a realizar saltando directamente al socket, pues el agente sabe el formato en el que se tienen que mandar los comandos y finalmente el socket envía la información al servidor. Este proceso ejemplifica la elección de la arquitectura, pues en una arquitectura por capas cerrada el agente tendría que comunicarse forzosamente con el intérprete para mandar el mensaje al socket.

Clases

A continuación se listan las clases creadas con una descripción breve de la responsabilidad de cada una.

- **MySocket.**
  - Envía y recibe mensajes desde el servidor a través de un socket UDP.
- **Interpreter.**

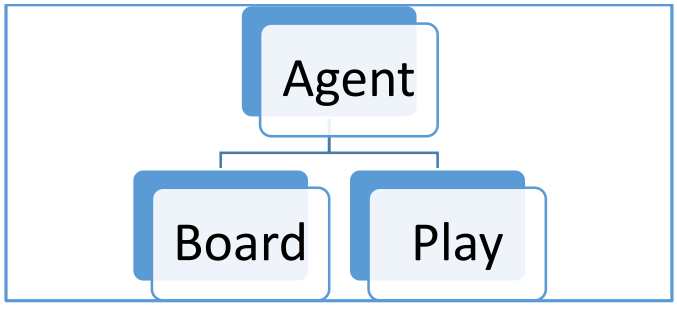
    - Decodifica el mensaje recibido por el servidor, convirtiéndolo en objetos fáciles de utilizar por el sistema.
- **Agent.**
    - Mantiene el estado del tablero y genera jugadas para mandar al servidor.
- **Board.**
    - Contiene información específica acerca del tablero necesaria para decidir las jugadas.
- **Play.**
    - Contiene información sobre una jugada específica para facilitar su manejo dentro del sistema.
- **OptParse.**
    - Permite leer las opciones pasadas al sistema mediante la línea de comandos.
- **MyLogger.**
    - Permite mantener un log de actividades para el análisis y/o depuración del sistema.
- **MultiIO.**
    - Permite mantener el log tanto en diversos dispositivos de salida con facilidad.
- **Core_ext.**
    - Añade funcionalidad a la biblioteca estándar de ruby.
- **Rain.**
    - Permite notificar de manera agradable al usuario que el agente ha ganado el juego
- **Client.**
    - Sincroniza los componentes y permite dirigir el flujo de datos entre diferentes ciclos del juego.

Subsistemas

El cliente está organizado de manera tal que responsabilidades parecidas pertenezcan a un mismo subsistema.

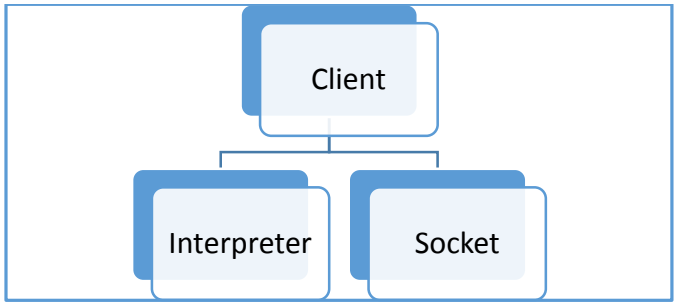
Generación de jugadas

Cuya responsabilidad es asegurarse de que se envíe la mejor jugada posible en cada ciclo del juego, este subsistema se ve reflejada en el código mediante una asociación, en la que el agente utiliza objetos de las clases Board y Play para realizar su función.

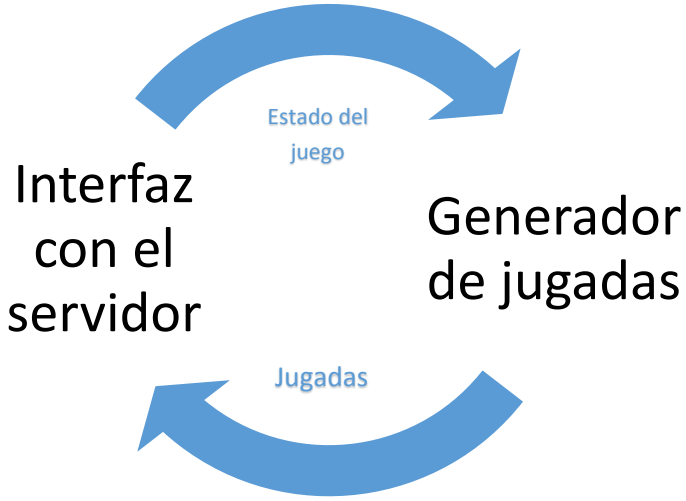


Interfaz con el servidor

En este, el cliente utiliza al intérprete para determinar qué tipo de mensajes espera el servidor y después utiliza al socket para enviar dichos mensajes.



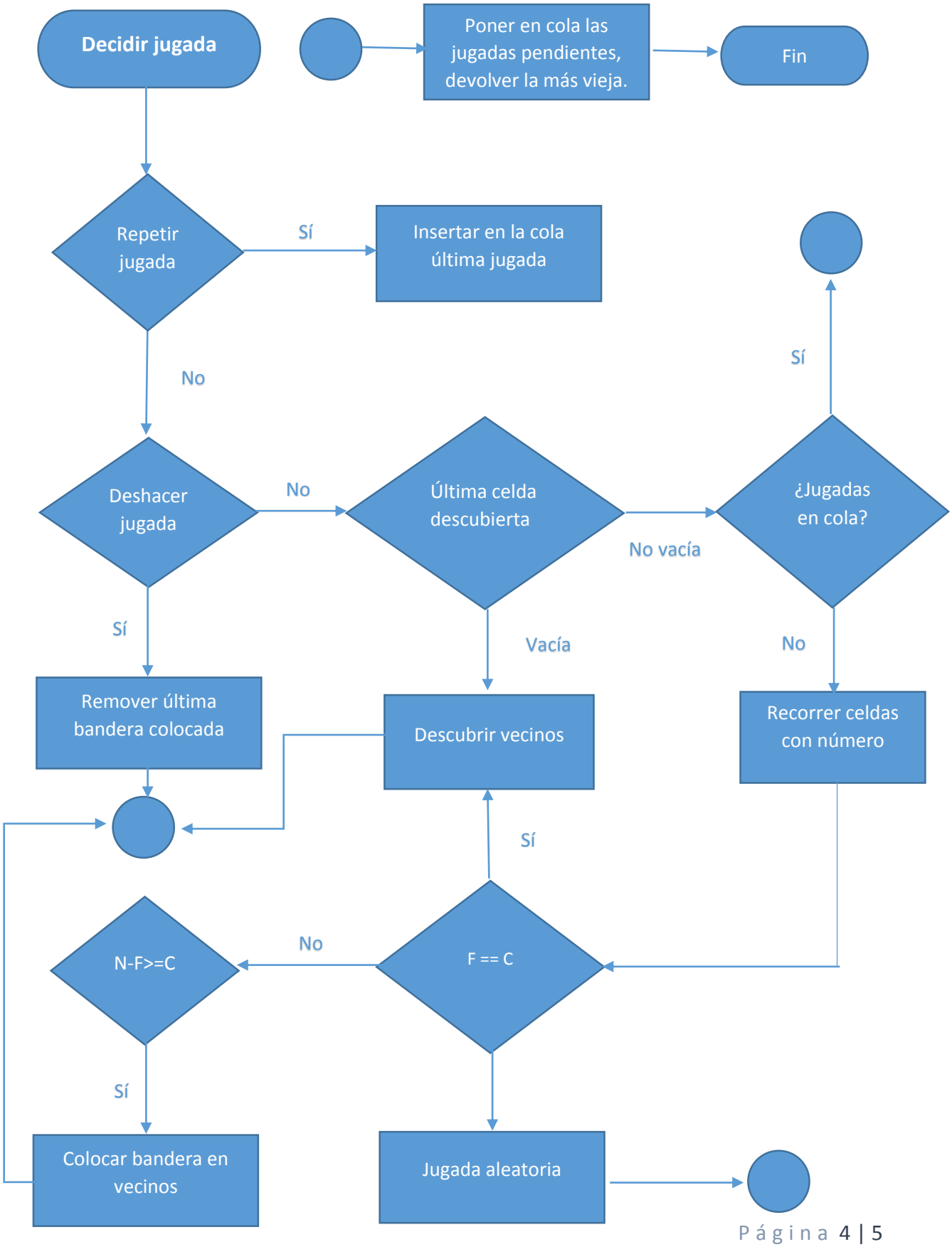
Interacción entre ambos



Agente

El agente es sin duda la parte crucial del sistema, pues en él se decide si se cumple el objetivo o no. Un ciclo normal, para ayudar a entender el proceso que sigue el agente para tomar una decisión analizaremos un diagrama de flujo bajo la siguiente notación.

- N es el número que tiene la celda actual.
- F es el número de celdas vecinas que tienen una bandera.
- C es el número de celdas vecinas cubiertas.


Decidir jugada
Poner en cola las jugadas pendientes, devolver la más vieja.
Fin
Repetir jugada
Sí
Insertar en la cola última jugada
No
Deshacer jugada
No
Última celda descubierta
No vacía
¿Jugadas en cola?
Sí
Sí
Vacía
No
Remover última bandera colocada
Descubrir vecinos
Recorrer celdas con número
Sí
N-F>=C
No
F == C
Sí
Colocar bandera en vecinos
Jugada aleatoria

Implementación

Desarrollado en lenguaje Ruby versión 1.9.3, documentado con YARD 0.8 y especificado con RSpec 2.14 utilizando Test Driven Development, en el cual, primero se escriben las especificaciones del código (pruebas) y después se escribe el código necesario para satisfacer dicha especificación. Este proceso se repite hasta haber alcanzado un objetivo de funcionalidad y, finalmente, se realiza un proceso de *refactoring* del código, proceso facilitado por la existencia de especificaciones ejecutables.

Documentación del código

Abrir el archivo index.html en la carpeta **./doc.** Este contiene un conjunto de archivos html que contienen los detalles de implementación del código.

Especificación

Se requiere instalar la gema Rspec, mediante el comando $ $gem\ install\ RSpec$ para poder ejecutar $ $rspec\ ./src/spec\ \ -f\ d\ -c$ y comprobar que el funcionamiento es el esperado.

Ejecución

Se requiere Ruby 1.9.3 o superior y permisos de ejecución al archivo **./minesweeperclient.rb** para ejecutar el script principal.

Los detalles de ejecución se encuentran en el archivo léeme.