Curso 2024/25

Tecnologías de la Información

Universidad de Huelva

GRADO EN INGENIERÍA INFORMÁTICA

ESTRUCTURAS DE DATOS I

Práctica 2
Estructuras dinámicas

Enunciado

Una compañía de videojuegos quiere montar una nueva infraestructura de servidores dedicados para sus nuevos títulos multijugador online. Como primer paso, el equipo responsable de este nuevo proyecto nos propone en esta práctica desarrollar de una aplicación C*++* que simule el funcionamiento de un sistema software para la gestión de dicha infraestructura.

A grandes rasgos, esta aplicación permitirá desplegar un número indefinido de servidores de juegos, realizando una configuración básica de los mismos. Una vez desplegados, permitirá simular la conexión de jugadores a los distintos servidores, e incluso gestionar colas de espera cuando éstos están completamente llenos.

De acuerdo con el documento de diseño proporcionado por la compañía de videojuegos, cada servidor será aproximado como un objeto de la clase *Servidor*, cuya declaración es la que sigue:

```
class Servidor{
    cadena direccionServidor; //direccion IP/hostname del servidor de juegos.
    cadena nombreJuego; //nombre del juego jugado en el servidor.
    int id; //código numérico (entero positivo) utilizado en el ámbito interno de la compañía para
            //identificar de forma unívoca cada servidor que ésta gestiona.
    Servidor *siguienteServidor; //puntero al siguiente servidor del sistema.
    int maxJugadoresConectados; //número máximo de jugadores que pueden ser alojados en el servidor.
    int maxJugadoresEnEspera; //número máximo de jugadores que pueden estar en espera para acceder al
                              //servidor.
    lista jugadoresConectados; //lista de jugadores alojados en el servidor. Los jugadores aparecen
                               //ordenados en la lista según su puntuación, y de forma ascendente en
                               //relación al valor de ésta.
    cola jugadoresEnEspera; //cola de jugadores en espera de poder acceder al servidor.

    cadena estado; //estado actual del servidor. Para este atributo sólo son posibles los valores que
                   //siguen: ACTIVO, INACTIVO o MANTENIMIENTO.
    int puerto; //número del puerto de escucha del servidor;
    cadena localizacionGeografica; //país en el que se encuentra ubicado físicamente el servidor.

public:
    ~Servidor();

    Servidor(cadena dS, cadena nJ, int i, int mxL, int mxC, int p, cadena lG);
    //método constructor que inicializa los atributos direccionServidor, nombreJuego, id,
    //maxJugadoresConectados, maxJugadoresEnEspera, puerto y localizacionGeografica con,
    //respectivamente, dS, nJ, i, mxL, mxC, p y lG. Además, inicializa el atributo estado a “INACTIVO”.

    int getId();
    //devuelve el valor del atributo id.

    void getDireccionServidor(cadena dS);
    //devuelve el valor de direccionServidor a través del parámetro de entrada/salida dS.

    void setSiguienteServidor(Servidor *pS);
    //asigna el valor recibido como parámetro al atributo siguienteServidor.

    Servidor* getSiguienteServidor();
    //devuelve el valor del atributo siguienteServidor.

    bool conectarJugador(Jugador j);
    //aloja al jugador j en el servidor, esto es, lo añade a la lista listaConectados siempre y cuando
    //no se haya alcanzado el número máximo de jugadores. Devolverá true si el jugador finalmente es
    //alojado en el servidor; false en caso contrario. Los jugadores se almacenan en la lista por
    //puntuación ascendente, por lo que el jugador deberá ser añadido a aquella posición para la que
    //se garantice el cumplimiento de esta regla de negocio.
```

```
bool ponerJugadorEnEspera(Jugador j);
//encola al jugador j en la cola de espera jugadoresEnEspera, comprobando previamente que el servidor
//ha alcanzado el número máximo de conexiones, y de que por otra parte no se ha alcanzado el número
//máximo de jugadores que pueden estar en espera para acceder al servidor. Devolverá true en caso de
//enconlar exitosamente al jugador; false en caso contrario.

void mostrarJugadoresConectados();
//muestra por pantalla el nombre, el identificador numérico, el ping(latencia), la puntuación global
//y el país desde el que se conecta cada uno de los jugadores alojados en el servidor.

void mostrarJugadoresEnEspera();
//muestra por pantalla el nombre, el identificador numérico, el ping(latencia) y la puntuación global
//y el país desde el que se conecta cada uno de los jugadores que se encuentran esperando poder
//acceder al servidor.

bool estaActivo();
//devuelve true si el estado del servidor es ACTIVO; false en caso contrario.

bool activar(); //activa un servidor en estado INACTIVO o MANTENIMIENTO. Si el servidor ya estaba
//activado, el método devolverá false; true en caso contrario, confirmando así que el servidor ha
//sido activado.

bool desactivar();
//desactiva un servidor en estado ACTIVO o MANTENIMIENTO. Si el servidor ya estaba desactivado, el
//método devolverá false; true en caso contrario, confirmando así que el servidor ha sido desactivado.
//Si el servidor a desactivar está previamente activo, será necesario vaciar tanto la lista de
//jugadores conectados como la cola de aquellos que están esperando poder conectarse.

bool ponerEnMantenimiento();
//pone un servidor en MANTENIMIENTO. Sólo es posible pasar al estado MANTENIMIENTO para aquellos
//servidores que están INACTIVOS. El método devolverá true en caso de poder poner al servidor en
//MANTENIMIENTO; false en caso contrario, esto es, cuando el servidor está previamente ACTIVO, o bien
//ya está en MANTENIMIENTO.

void mostrarInformacion();
//muestra por pantalla información de interés del servidor. En concreto, muestra la dirección y el
//identificador del servidor, su estado, la relación entre el máximo de jugadores que puede alojar y
//el número real de jugadores conectados. También mostrará el número máximo de jugadores que pueden
//estar esperando acceso, junto con el número de elementos tipo Jugador encolados en jugadoresEnEspera
//con dicho propósito. Finalmente, también mostrará el puerto de escucha del servidor y la latencia
//media de los jugadores conectados, y la localización geográfica del servidor (país).

bool expulsarJugador(cadena nombre);
//expulsa el jugador cuyo nombreJugador coincide con el parámetro nombre de la lista de conectados,
//o bien de la cola de espera, en función de dónde esté contenida la instancia de tipo Jugador
//correspondiente. Si el jugador es localizado en alguna de estas estructuras y convenientemente
//eliminado el método devolverá true; false en caso contrario. Si el jugador a expulsar es localizado
//en la lista de jugadores conectados, tras eliminarlo, será necesario comprobar si hay alguien en
//la lista de espera. En caso positivo, el primero de la cola será automáticamente conectado
//al servidor, pasando a la lista de conectados.

void getNombreJuego(cadena nJ);
//devuelve el valor del atributo nombreJuego a través del parámetro de entrada/salida nJ.

int getPuerto();
//devuelve el valor del atributo puerto.

void getLocalizacionGeografica(cadena lG);
//devuelve el valor del atributo localizacionGeografica a través del parámetro de entrada/salida lG.

int getMaxJugadoresConectados();
//devuelve el valor del atributo maxJugadoresConectados.

int getMaxJugadoresEnEspera();
//devuelve el valor del atributo maxJugadoresEnEspera.
```

```
    int getNumJugadoresConectados();
    //devuelve el número de jugadores conectados al servidor.

    int getNumJugadoresEnEspera();
    //devuelve el número de jugadores en la cola de espera de acceso al servidor.

    void exportarJugadoresConectados(Jugador *conectados);
    //el método copiara todos los jugadores alojados en el servidor en el vector dinámico accesible desde
    //el puntero conectados, parámetro de entrada.

    void exportarJugadoresEnEspera(Jugador *enEspera);
    //el método copiara todos los jugadores que están en la cola de espera para el acceso al servidor en
    //el vector dinámico accesible desde el puntero conectados, parámetro de entrada.
};
```

Dónde las clases *lista* y *cola* son respectivamente implementaciones dinámicas de los TADs lista y cola que almacenan elementos del tipo struct *Jugador*, declarado como sigue:

```
struct Jugador{
    cadena nombreJugador; //nombre de usuario del un jugador. Es único en el sistema.
    int ID; //código numérico (entero positivo) identificador de un jugador. Es único en el sistema.
    bool activo; //valor lógico que indica si el jugador está o no conectado a un servidor.
    int latencia; //tiempo de respuesta del jugador calculado por el sistema cuando recibe una petición
                  //de conexión. Se mide en milisegundos.
    long puntuacion; //puntuación acumulada por el jugador a lo largo de todas las temporadas en las que
                     //éste ha participado en el juego
    cadena pais; //nombre del país desde el que se conecta el jugador.
};
```

El tipo ad-hoc *cadena* aproxima una cadena de caracteres a partir de un vector de elementos tipo *char*. Su definición es la siguiente:

```
typedef char cadena[50];
```

Por otra parte, para gestionar el conjunto de servidores de juegos desplegados por la compañía se ha convenido el desarrollo de una clase que permitirá gobernar esta infraestructura en su conjunto. La clase, de nombre *GestorServidores*, tiene la siguiente declaración:

```
class GestorServidores{

    Servidor *primerServidor; //puntero al primer elemento/servidor dentro de la estructura de nodos
    //enlazados/objetos de la clase Servidor, representativa del conjunto de servidores utilizados para
    //gestionar las partidas multijugador online de los distintos juegos de la compañía.

    int numServidores; //número de servidores de juego desplegados gobernados por el gestor.

public:
    GestorServidores();
    //método constructor que inicializará convenientemente los atributos de la clase a los valores por
    //defecto convenidos.

    ~GestorServidores();
    //método destructor que liberará la memoria reservada para los artefactos dinámicos que integran
    //la estructura de nodos enlazados correspondiente al conjunto de servidores.

    Int getNumServidores();
    //devuelve el valor del atributo numServidores.
```

```
bool desplegarServidor(cadena dS, canedena nJ int i, int mxL, int mxC, int p, cadena lG);
//desplegará un nuevo servidor de juego en el sistema. Para ello, creará un objeto de la clase
//Servidor, cuya dirección/hostname será dS, el juego ejecutado nJ, su identificador i, el máximo de
//jugadores que puede alojar será mxL, y el máximo de jugadores que pueden estar en espera de conexión
//mxC; su puerto de escucha será p, y el nombre del país en donde está físicamente instalado lG.
//Este nuevo objeto será integrado en la estructura del nodos, y su estado por defecto será INACTIVO.
//Estos nodos están ordenados por orden alfabético ascendente según el nombre del país en el que cada
//servidor está ubicado (atributo localizacionGeografica). El nuevo nodo será añadido en dicha
//estructura respetando este requisito. El método devolverá true si el despliegue del nuevo nodo es
//completado con éxito; false en caso contrario, bien porque no haya sido posible reservar memoria
//para crear el nuevo objeto, o bien porque ya existiese en la estructura de nodos otro Servidor con
//el mismo nombre, o bien el mismo identificador que el nuevo elemento a integrar en la estructura.

bool desconetarServidor(cadena dS);
//el método intentará desconectar (poner en estado INACTIVO) el servidor cuya dirección/hostname
//coincide con el valor indicado en el parámetro de entrada dS. Si el servidor a desactivar estaba
//ACTIVO, será necesario expulsar a los jugadores que estaban en espera, y redirigir a los que estaban
//ya alojados en el servidor a otros nodos activos del sistema con el mismo juego instalado. El
//proceso de redistribución de los jugadores priorizará aquellos con menor tiempo de respuesta (ping).
//La distribución se realizará de forma equitativa entre los distintos servidores activos del sistema,
//esto es, cada jugador considerado será alojado en el servidor con el mayor número de espacios
//disponibles en su correspondiente lista de jugadores conectados. El mismo criterio se seguirá en
//caso de no poder alojar al jugador que toca al alguno de los servidores al estar todos llenos, y
//tener que incluirlo en alguna cola de espera. Aquellos jugadores para los que no haya espacios
//disponibles en alguno de los servidores, ni en ninguna de las colas de espera, serán expulsados
//del sistema.

bool conectarServidor(cadena dS);
//el método permite activas (poner en estado ACTIVO) el servidor cuya dirección/hostname coincide
//con el valor indicado en el parámetro de entrada dS. Devolverá true en caso de activar exitosamente
//el servidor indicado; false en caso contrario, bien porque el servidor ya estaba activo, o bien
//porque directamente no existía ningún nodo con la dirección/hostname indicado.

bool realizarMantenimiento(cadena dS);
//el método permite poner en estado MANTENIMIENTO el servidor cuya dirección/hostname coincide con
//el valor indicado en el parámetro de entrada dS. Devolverá true en caso de poner exitosamente el
//servidor indicado en mantenimiento; false en caso contrario, bien porque el servidor ya estaba en
//dicho estado, o bien porque directamente no existía ningún nodo con la dirección/hostname indicado.

bool eliminarServidor(cadena dS);
//el método eliminará de la estructura de nodos enlazados el servidor cuya dirección/hostname coincide
//con el valor indicado en el parámetro de entrada dS. Este método es aplicable solamente sobre
//aquellos servidores en estado INACTIVO o MANTENIMIENTO. El método devolverá true en caso de
//eliminarse de forma exitosa el servidor indicado; falso en caso contrario, esto es, cuando no
//exista ningún nodo con la dirección/hostname indicado, o bien el servidor afectado no esté INACTIVO
//o en MANTENIMIENTO.

bool alojarJugador(Jugador j, cadena nomJuego, cadena host, bool &enEspera);
//el método intentará alojar al jugador (tipo Jugador) j en algún servidor ACTIVO para el juego de
//nombre nomJuego. Si no se encuentra ningún servidor activo del juego indicado, el método finalizará
//devolviendo false, además de otro false a través del parámetro por referencia enEspera. En caso
//contrario, criterio será el de alojar al jugador en el servidor cuya diferencia entre el número
//máximo de jugadores soportado y el número de usuarios actualmente conectados sea la mayor. Si se
//encuentra algún servidor con conexiones disponible el jugador será alojado en éste añadiéndolo a
//la lista jugadoresConectados correspondiente. El método finalizará devolviendo true, junto con la
//dirección/hostname del servidor a través del parámetro de entrada/salida host. Si por contra no
//hay espacio disponible en ninguno de los servidores, se intentará añadir al jugador a alguna de
//las colas de espera. El criterio será el de añadir al jugador a aquella cola para la que la
//diferencia entre el número máximo de jugadores que pueden estar en espera, y el número de elementos
//ya encolados es la mayor. En caso de encolar al jugador en alguna cola de espera, el método
//devolverá un false, además de un true a través del parámetro por referencia enEspera, indicando
//así que el jugador no ha podido ser alojado en alguno de los servidores, y que se encuentra a la
//espera de poder conectarse a alguno. Además, en ese caso se indicará el nombre/dirección del
//servidor dónde está encolado a través del parámetro de entrada/salida host. Si el jugador no ha
//podido ser conectado ni encolado en una estructura de espera, el método finalizará nuevamente
//devolviendo un false, junto con otro false a través del parámetro por referencia enEspera.
```

```
    bool expulsarJugador(cadena nJ, cadena host);
    //el método expulsará al jugador de nombre nJ de cualquier servidor o cola de espera del sistema
    //en la que esté ubicado. El método devolverá true en caso de localizar y expulsar al jugador
    //indicado, devolviendo además la dirección/hostname del servidor en el que estaba conectado o bien
    //a la espera de acceso al mismo; false en caso contrario, al no encontrarse en el sistema el jugador
    //indicado. Si el jugador en cuestión estaba conectado a un servidor, al ser expulsado habrá que
    //garantizar que, en caso de haber jugadores en cola de espera para acceder al mismo, el primero de
    //dicha estructura pase a ser alojado en este servidor de forma automática.

    int getPosicionServidor(cadena dS);
    //el método devolverá la posición en la que se encuentra el servidor cuya dirección/hostname es igual
    //a dS dentro de la secuencia que forman el conjunto de nodos enlazados, siendo el primer nodo el
    //que ocupa la posición 1. Si no hay en la estructura de nodos enlazados ningún servidor con la
    //dirección indicada, el método devolverá un -1.

    void mostrarInformacionServidores(int pos);
    //muestra por pantalla información del servidor que se encuentra en la posición indicada por el
    //parámetro pos dentro de la secuencia de nodos-servidores, dónde el primer servidor es el situado
    //en la posición 1. Si la posición indicada está fuera de rango, deberá informarse al usuario de tal
    //circunstancia a través de un mensaje de error. Si la posición indicada a través de pos es -1,
    //deberá mostrarse por pantalla información sobre la totalidad de los servidores contenidos en la
    //estructura de nodos enlazados. Para servidores activos, será también necesario mostrar por pantalla
    //los datos de los jugadores conectados, así como la de aquellos que estén en la cola de acceso al
    //mismo.

    bool jugadorConectado(cadena nJ, cadena dS);
    //el método devolverá true si el jugador con nombre nJ está conectado al servidor con
    //dirección/hostname dS; false en caso contrario.

    bool jugadorEnEspera(cadena nJ, cadena dS);
    //el método devolverá true si el jugador con nombre nJ está en la cola de espera del servidor con
    //dirección/hostname dS; false en caso contrario.

    //el método devolverá true si el jugador con nombre nJ está conectado a alguno de los servidores
    //activos del sistema.
    bool jugadorConectado(cadena nJ);

    //el método devolverá true si el jugador con nombre nJ está en la cola de espera de alguno de los
    //servidores activos del sistema.
    bool jugadorEnEspera(cadena nJ);

};
```

El conjunto de servidores (objetos de la clase *Servidor*) gobernados por un objeto de la clase *GestorServidores* se estructura como un conjunto de nodos enlazados, de forma que cada servidor guarda en su atributo *siguienteServidor* la dirección al siguiente nodo dentro de la secuencia. El primer nodo dentro de la misma será accesible a través del puntero *primerServidor*, atributo de la clase *GestorServidores*. En esta línea, el último servidor de este conjunto de nodos será identificable por almacenar un valor nulo (*NULL*) en su atributo *siguienteServidor*. En el siguiente diagrama, a modo de resumen, se muestra cómo se organizaría una estructura de nodos controlados por un objeto de la clase *GestorServidores*:

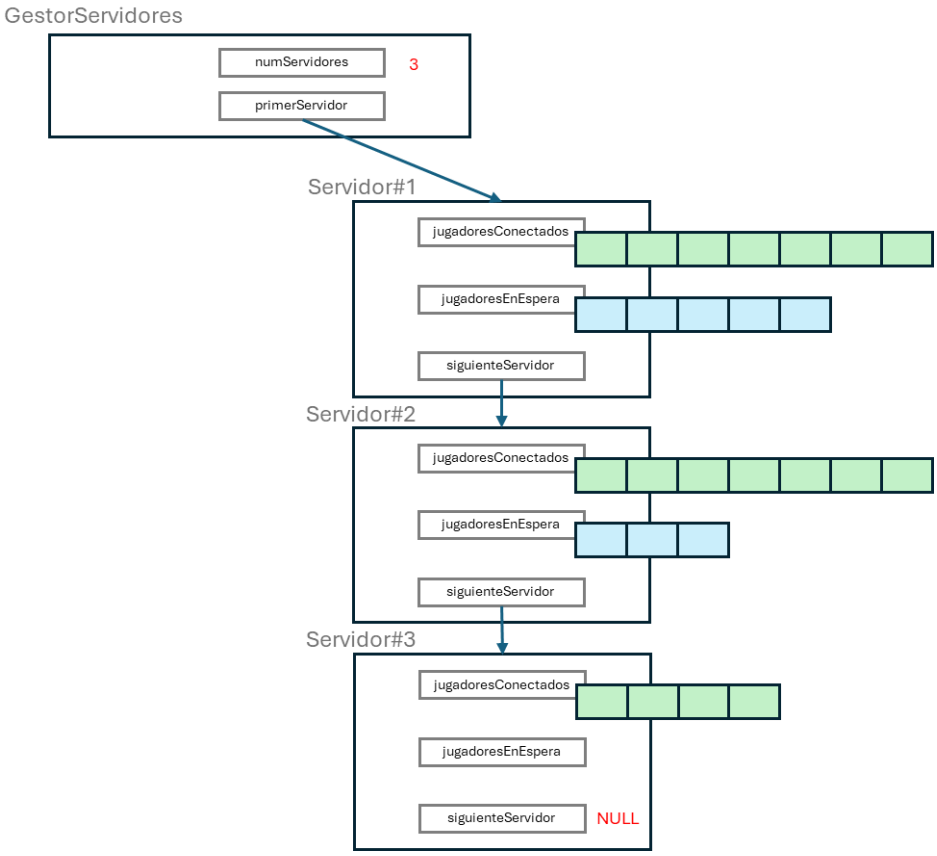

Figura 1.- Representación de la estructura de nodos enlazados correspondiente a un conjunto de tres servidores de juego gestionados por un objeto de la clase *GestorServidores*.

De acuerdo con las definiciones y clases propuestas, se pide el desarrollo de una aplicación que a partir de la declaración de único objeto de la clase *GestorServidores* permita desplegar, gestionar y eliminar distintos servidores de juego. Una vez arrancada la aplicación, se mostrará un menú como el propuesto a continuación, desde el cual se podrá acceder a las distintas opciones del programa:

```
GESTOR DE SERVIDORES v1.0
================================================================
1. Mostrar servidor.
2. Crear servidor.
3. Eliminar servidor.
4. Activar servidor.
5. Desactivar servidor.
6. Programar mantenimiento de servidor.
7. Conectar jugador.
8. Expulsar jugador.
9. Salir.

> Seleccione una opción:
```

Figura 2.- Menú principal de la aplicación a desarrollar.

A continuación, se describe en detalle cada una de estas opciones:

1. ***Mostrar servidor.*** A través de esta opción, será posible mostrar por pantalla toda la información referida al servidor de juego indicado por el usuario de la aplicación tras introducir su dirección/hostname. Para el servidor indicado, se mostrará su código de identificación y dirección, el número de puerto de escucha, el país dónde está físicamente instalado, el número máximo de conexiones que acepta, y finalmente su estado. Si éste último es *ACTIVO*, será necesario además listar todos los jugadores alojados en el servidor, así como aquellos que puedan estar en espera de acceder al mismo. Si no existe ningún servidor con la dirección indicada, será necesario informar al usuario de tal circunstancia con un mensaje de error. Si el usuario introduce la palabra reservada *ALL*, se mostrará por pantalla la información correspondiente a la totalidad de servidores desplegados en el sistema.
2. ***Crear servidor.*** A través de esta opción se desplegará un nuevo servidor en el sistema. El usuario deberá introducir por teclado los siguientes datos con los que inicializar el nuevo servidor: dirección del servidor, código identificador, nombre del juego instalado en el servidor, número de puerto de escucha, número máximo de conexiones aceptadas y número máximo de jugadores en espera soportados, y su localización geográfica (nombre del país). Previo despliegue, será necesario comprobar que no existe ya ningún otro servidor con la dirección o el identificador indicados por el usuario de la aplicación. Si se diese esta situación, evidentemente no se creará el nuevo servidor, notificándose al usuario de esta circunstancia mediante un mensaje de error.
3. ***Eliminar servidor.*** Con esta opción será eliminado del sistema el servidor cuya dirección/hostname coincida con el indicado por el usuario. Antes de ser eliminado, internamente el servidor en cuestión deberá ser desactivado, redistribuyendo a aquellos jugadores alojados en el mismo en caso de que éste estuviese activo. El criterio de redistribución es el establecido en la explicación del método *GestorServidores::desconectarServidor*. Si no hay ningún servidor con dirección coincidente a la indicada por el usuario, éste será notificado de este hecho con un mensaje de error.
4. ***Activar servidor.*** A través de esta opción será posible activar el servidor de dirección/hostname el indicado por el usuario. Si este servidor estuviese ya activo, el usuario será notificado de dicha circunstancia con un mensaje de error. Igualmente será necesario informar con un mensaje de error al usuario en caso de que el servidor indicado por éste no exista en el sistema.
5. ***Desactivar servidor.*** A través de esta opción será posible desactivar el servidor de dirección/hostname el indicado por el usuario. Si este servidor estuviese ya desactivado, el usuario será notificado de dicha circunstancia con un mensaje de error. Igualmente será necesario informar con un mensaje de error al usuario en caso de que el servidor indicado por éste no exista en el sistema. Si el servidor en cuestión estuviese previamente *ACTIVO*, será necesario redistribuir en otros nodos a aquellos jugadores conectados al mismo. Nuevamente, El criterio de redistribución es el establecido en la explicación del método *GestorServidores::desconectarServidor*.
6. ***Programar mantenimiento de servidor.*** Esta opción permite poner el servidor cuya dirección coincide con la indicada por el usuario en estado *MANTENIMIENTO*. Esto es sólo posible si el servidor se encuentra previamente en estado *INACTIVO*. Por tanto, habrá que notificar al usuario con un mensaje de error en caso de no poderse realizar el cambio de estado por este motivo. Igualmente será necesario informar con un mensaje de error al usuario en caso de que el servidor indicado por éste no exista en el sistema.

7. ***Conectar jugador.*** Esta opción permitirá conectar a un jugador en un servidor desplegado para el juego indicado. Así, en primera instancia la aplicación pedirá que el usuario introduzca por teclado el nombre (*nick*) del jugador. Seguidamente la aplicación comprobará en primer lugar que dicho jugador no está ya alojado en algún servidor. Si se verifica que el jugador ya está conectado al sistema, se

notificará mediante un mensaje de error de esta circunstancia y el proceso finalizará. En caso contrario, se pedirá al usuario que introduzca los siguientes datos: identificador único del jugador, nombre del país desde el que se conecta y nombre del juego al que pretende jugar. Los valores de *latencia* y *puntuacion* deberán generarse de forma aleatoria. Para el tiempo de respuesta, será necesario generar un valor entero comprendido entre 1 y 500, ambos incluidos; para la puntuación, un valor entre 0 y 99999, también ambos incluidos. Tras esto, se intentará conectar al jugador a algún servidor del juego indicado, siguiendo el criterio establecido en la explicación del método *GestorServidores::alojarJugador*. Será necesario confirmar por pantalla cual ha sido el resultado del proceso, en base a tres distintas posibilidades:

a. El jugador ha sido conectado a un servidor. En este caso, habrá que notificar este hecho e indicar la drección/hostname del servidor en el que ha sido alojado.
b. El jugador está a la espera de acceder a un servidor. En este caso, habrá que notificar este hecho e indicar la drección/hostname del servidor en el está a la espera de ser alojado.
c. El jugador no ha podido ser conectado/encolado en una cola de espera al no encontrarse ningún servidor activo para el juego indicado con espacio disponible. En este caso será necesario mostrar un mensaje de error por pantalla para informar al usuario de la aplicación.

8. ***Expulsar jugador.*** Esta opción permite expulsar del sistema al jugador cuyo nombre (*nick*) coincida con el indicado por el usuario. Si el jugador indicado no se encuentra en el sistema habrá que notificar dicha circunstancia a través de un mensaje de error por pantalla. En caso contrario, se confirmará con un mensaje el éxito de la operación de la expulsión, mostrando además la dirección/hostname del servidor del que el jugador fue eliminado. Además, si el jugador estaba conectado (no en espera) en un servidor, tras ser expulsado de éste, habrá que comprobar si hay alguien a la espera de acceder al mismo. En caso positivo, este nuevo jugador abandonará la cola de espera en la que estaba ubicado para conectarse definitivamente al servidor.
9. ***Salir.*** Permite salir de la aplicación.

Notas:

- Está permitida la implementación de métodos privados auxiliares para las clases *Servidor* y *GestorServidores*, pero no la inclusión de nuevos atributos, o bien la modificación de los propuestos en el enunciado.
- Las implementaciones de las clases *lista* y *cola*, que almacenarán elementos de tipo Jugador, deberán estar basadas en las implementaciones dinámicas para dichos TADs propuestas en el tema 4 de la parte teórica de la asignatura.