Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería en Sistemas

Programación de Redes

Obligatorio 1

Itai Miller - 201244

Rafael Alonso - 201523

Grupo M6A
Docente: Luis Barrague

Índice

1. Introducción

El sistema consiste en un proyecto consola con dos aplicaciones, una para el cliente y otra servidor. Entre las dos forman un sistema de juego en el que el servidor maneja las partidas y varios clientes pueden registrarse y conectarse para jugar a la misma contra otros clientes conectados.

1.1 Autenticación

Para poder ingresar al sistema se requiere un nombre de usuario registrado. Un usuario solo puede estar logueado con el mismo nombre de usuario en una sola sesión, es decir dos clientes no pueden loguearse al mismo usurario. Si cierra sesión, se pierde la conexión o cierra el programa, este usuario será liberado para que otro cliente pueda loguearse con el mismo.

1.2 Partida

Para poder jugar una partida se requiere de las siguientes condiciones:

- Registro de usuario.
- Loguearse.
- Unirse a la partida siempre y cuando en el servidor exista una, si no, se le informa de que debe esperar a que comience la partida.

1.3 Decisiones de diseño de la partida

- Antes de iniciar una partida, el servidor hace un broadcast avisándole a todos los clientes que la partida comienza en 10 segundos. Esto es para que los mismos se puedan ir uniendo.
- Decidimos que la partida tenga como máximo 32 jugadores ya que como el tablero es de 64 posiciones (por letra), se puedan mover atacar. Sentíamos que, si dejábamos entrar más jugadores, se iba a complicar el desplazamiento de los jugadores.
- Utilizamos el patrón Singleton en dos ocasiones para controlar de que exista una única instancia del mismo. Lo implementamos en la clase Match para controlar de que haya una única partida y también para la lista de jugadores hicimos una lista estática para controlar de que exista una única instancia de jugadores conectados.
- Decidimos que los jugadores al tablero se agreguen en una posición al azar donde no haya jugadores adyacentes que lo puedan atacar. Esto se debe gracias a que el límite de jugadores activos es 32 y el tablero es de 64 posiciones, esto es posible debido al principio del palomar.
- No consideramos a las posiciones diagonales como adyacentes.
- Si el movimiento o ataque que realiza el jugador es invalido, se le avisa que la acción fue invalida y se libera el turno.
- Cuando un jugador se mueve, le pide dos movimientos, sin embargo, uno de estos puede ser vacío, así realizando un solo movimiento en lugar de los dos.
- Si el segundo movimiento del jugador es invalido, el primer movimiento se mantiene realizado, pero se le avisa de que el segundo movimiento no fue posible y se ignora su segundo movimiento.
- La partida comienza cuando se ingresa en el servidor el comando para iniciar la partida.
- Si el cliente está jugando una partida y se desconecta o desloguea, inmediatamente su jugador muere.
- Si el cliente inserta un comando que requiere de otras especificaciones (registrarse, loguearse, seleccionar personaje y moverse), el comando no se procesara hasta que ingrese las otras especificaciones correctamente.
- El tiempo que dura la partida y el IP decidimos configurarlos en un archivo JSON para que puedan ser modificados una vez compilado el programa.

2. Interfaz

2.1 Usuario

La interfaz del usuario es por consola. En la misma, el usuario podrá realizar las siguientes acciones:

- Registrarse en el sistema.
- Loguearse en el sistema.
- Desloguearse del sistema.
- Unirse al juego.
- Unirse a la partida active.
- Seleccionar un rol (monstruo o sobreviviente).
- Realizar acciones durante la partida active (mover o atacar).
- Al finalizar el juego se muestran los resultados.
- Salir del programa.

En todo momento se le informa al usuario si tuvo éxito en su pedido o si tuvo algún error indicándole el mismo.

2.2 Servidor

El servidor es un proyecto de consola. Al iniciar, se muestra por consola "Listening" para indicar que ya puede aceptar conexiones.
En el mismo, se pueden realizar las siguientes acciones:

- Ver jugadores registrados.
- Ver jugadores logueados.
- Iniciar partida.
- Salir del programa.

Una vez iniciada la partida, no se puede ingresar ningún otro comando hasta que la misma termine.

3. Arquitectura

La solución está conformada por cuatro proyectos. Uno para el cliente (GameClient), otro para el servidor (GameServer), otro de Dominio que es usado por el servidor y el cuarto que es compartido por el cliente y el servidor (GameComm) que define los elementos esenciales de nuestro protocolo de conexión. El Cliente (GameClient) y el Servidor (GameServer) son completamente independientes entre sí, ya que se deben de poder conectar a través de computadoras completamente distintas. Para la comunicación entre ellos usamos sockets TCP lo cual permite enviar información entre estos.

3.1 GameClient

En este proyecto se encuentra todo lo que tiene que ver con los pedidos que le realiza el cliente al servidor para establecer una conexión. El GameClient consta con dos hilos paralelos, uno para manejar todos los comandos que el cliente quiere enviar al servidor y otro para recibir todos los mensajes del servidor. Esto se hizo para que no sean mutuamente exclusivos y el cliente pueda recibir mensajes importantes del servidor sin que el cliente haya ingresado ningún comando. También se encuentran los diferentes comandos y los mismos son los siguientes:

- Comando registro ("REGISTER").
- Comando loguearse ("LOGIN").

- Comando desloguearse (“LOGOUT”).
- Comando unirse al juego (“JOINMATCH”).
- Comando elegir jugador (“SELECTCHARACTER”)
- Comando moverse (“MOVE”).
- Comando atacar (“ATTACK”).
- Comando salir (“EXIT”).

3.2 GameServer

En este proyecto se encuentra todo lo que tiene que ver con los pedidos que atiende el servidor y las respuestas que brinda. El server consta de dos clases principales. La primera es ServerMain la cual se encarga de recibir todos los pedidos de conexión de clientes, crear sus instancias de handlers y de enviar todos los broadcasts importantes a todos los clientes en simultaneo. La otra clase importante es ClientHandler, esta es instanciada con hilos paralelos por ServerMain y cada instancia de ClientHandler se encarga de manejar todos los requests de cada cliente individualmente. A su vez el ServerMain maneja los comandos que ingresa el usuario en el servidor en un hilo por separado. Este mismo hilo maneja la instancia estática de la clase Match donde esta toda la lógica del juego y también tiene ciertos comandos que son:
- Comando mostrar jugadores registrados (“SHOWREGISTERED”).
- Comando mostrar jugadores logueados (“SHOWLOGGED”).
- Comando iniciar partida (“STARTMATCH”).
- Comando salir del juego (“EXIT”).

En el main de este proyecto, se aceptan todos los pedidos de conexión con un TcpListener que esta siempre pendiente a nuevas conexiones. En base a estos se crean instancias paralelas de las clases que manejan los comandos del cliente.

3.3 GameComm

En este proyecto se encuentran como dijimos anteriormente todos los elementos esenciales y requeridos por el cliente y servidor. Decidimos hacerlo por separado de ambos para reducir el acoplamiento, aumentar la cohesión y maximizar el reúso.
Estos son:
- Archivo de configuración de IP y tiempo de partida.
- Interfaz del intérprete de comandos.
- Las clases (una para pedidos y otra para respuestas) con todas las constantes de los comandos numerados para el protocolo de comunicación.
- La clase abstracta Comando (luego extendida por los comandos del cliente y servidor).
- La excepción que se usa en caso de desconexión.
- El texto correspondiente a cada uno de los comandos para escribirlos en la consola.

3.4 Dominio

En este proyecto se encuentran todas las entidades usadas por el servidor para manejar los datos de los usuarios y la partida.

4. Formato de mensajes

En la siguiente tabla se muestra la lista de comandos utilizados en la comunicación cliente – servidor con la información sobre su variable (en caso de ser usada).
Los CMD menores a 100 son pedidos y los CMD mayores a 100 son respuestas.

SOLICITUD	CMD	LARGO	VARIABLE
Registro de usuario	001	Variable	Nickname + imagen
Login	002	Variable	Nickname
Logout	003	00000	No
Unirse partida	004	00000	No
Seleccionar rol	005	Variable	Rol elegido
Moverse	006	1 o 2	Mov. 1 + Mov. 2
Atacar	007	00000	No
Salir	099	00000	No
Ok	100	Variable	Mensaje
Broadcast	198	Variable	Mensaje
Salir	199	00000	No
Registro inválido	101	00000	No
Login inválido	102	00000	No
Servidor lleno	103	00000	No
Desconocido	104	00000	No
No logueado	105	00000	No
Ya logueado	106	00000	No
Fin partida	109	00000	No
Partida llena	110	00000	No
No en la partida	111	00000	No
En partida	112	00000	No
Jugador muerto	113	00000	No
Fuera del límite	114	00000	No
Posición ocupada	115	00000	No
No eligió personaje	116	00000	No
Inválido mientras juega	120	00000	No
No jugando	121	00000	No

5. Monitores

Para manejar la concurrencia se utilizaron monitores.

Se utiliza objetos estaticos readonly genericos para controlar el acceso a las variables compartidas como la informacion de la partida o la lista de usuarios. Cada clase tiene su objeto readonly propio.

Esto se hace para manejar la consistencia de datos entre los distintos threads del servidor.

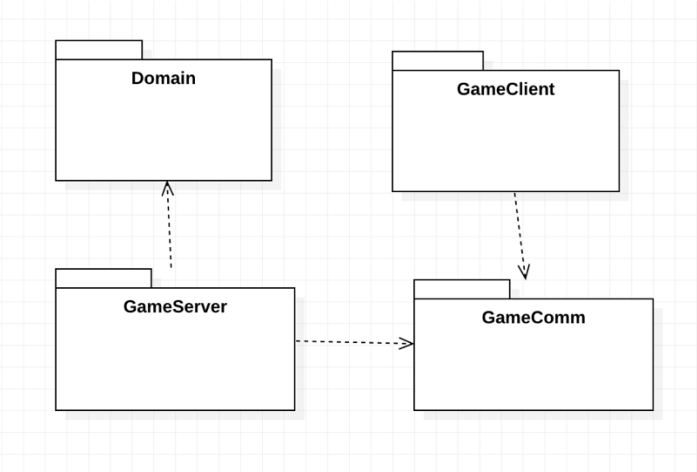
Para manejar la parte de los turnos decidimos hacerlo tambien con monitores para que dos jugadores no puedan realizar acciones de forma concurrente. La pila del lock de los monitores maneja el "orden" de los turnos, el cual se basa en el primero en realizar un pedido. Como los jugadores no pueden realizar otra accion hasta que el servidor responda, los monitores se encargan de la implementación de los comandos "tank".

No se utilizan monitores dentro de monitores y se controlaron bien los metodos con loops dentro de monitores para asegurarse de que se eviten deadlocks dentro del programa.

6. Diagramas

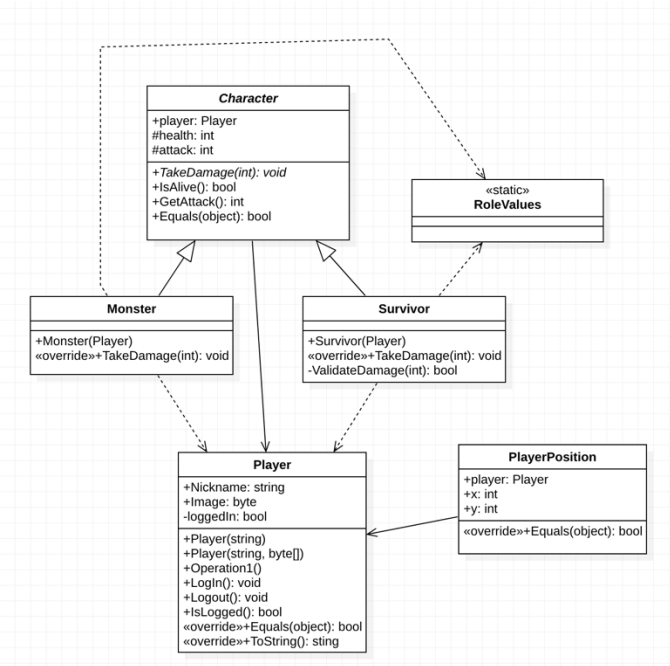
6.1 Diagrama de paquetes

En el siguiente diagrama se puede notar claramente como GameClient y GameServer usan a GameComm que es el proyecto compartido por ambos.



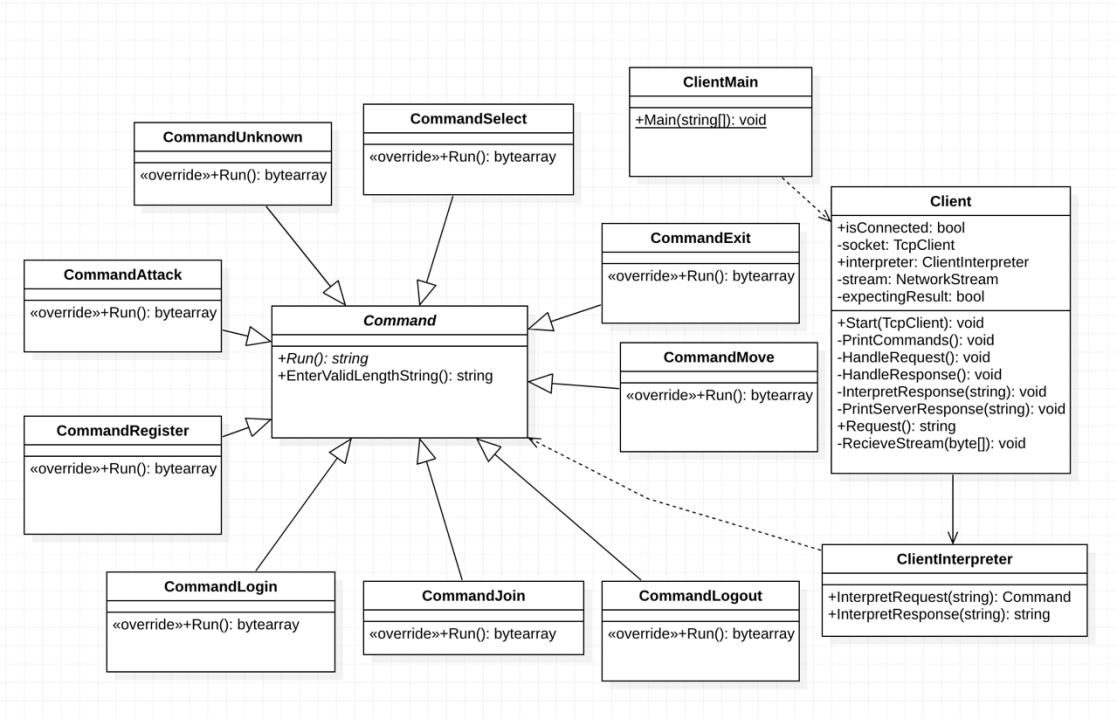
6.2 Diagrama de clases

6.2.1 Proyecto Domain

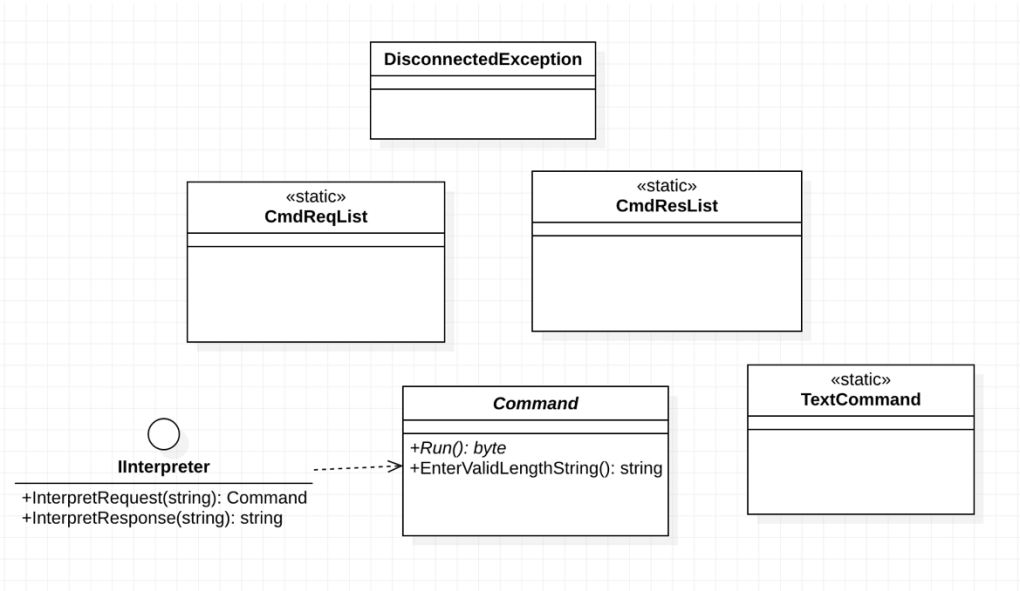


6.2.2 Proyecto GameClient

En las clases que heredan de Command se quiso poner que el método Run retorne byte[] pero como el StarUml no dejaba poner esa notación le pusimos bytearray.

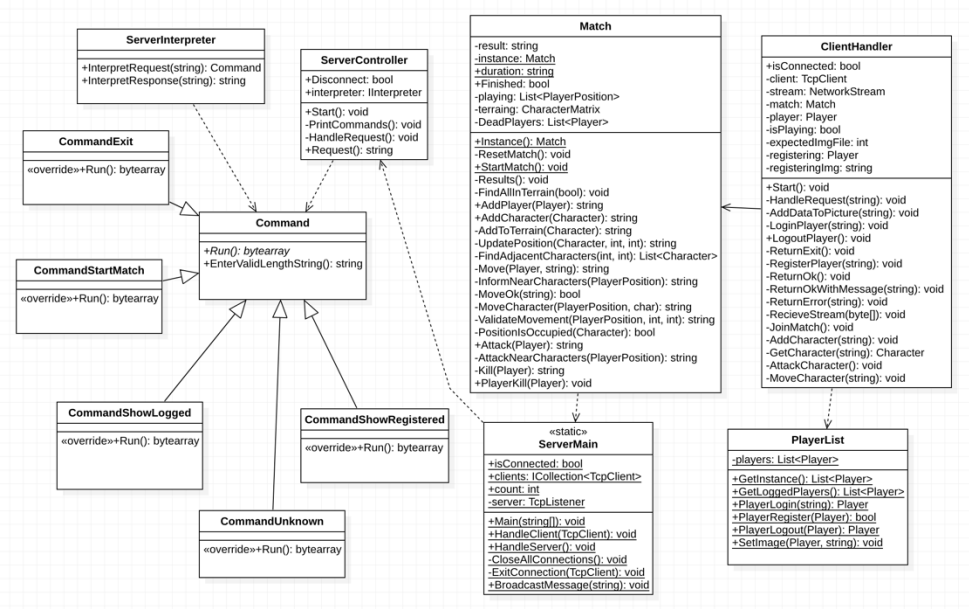


6.2.3 Proyecto GameComm



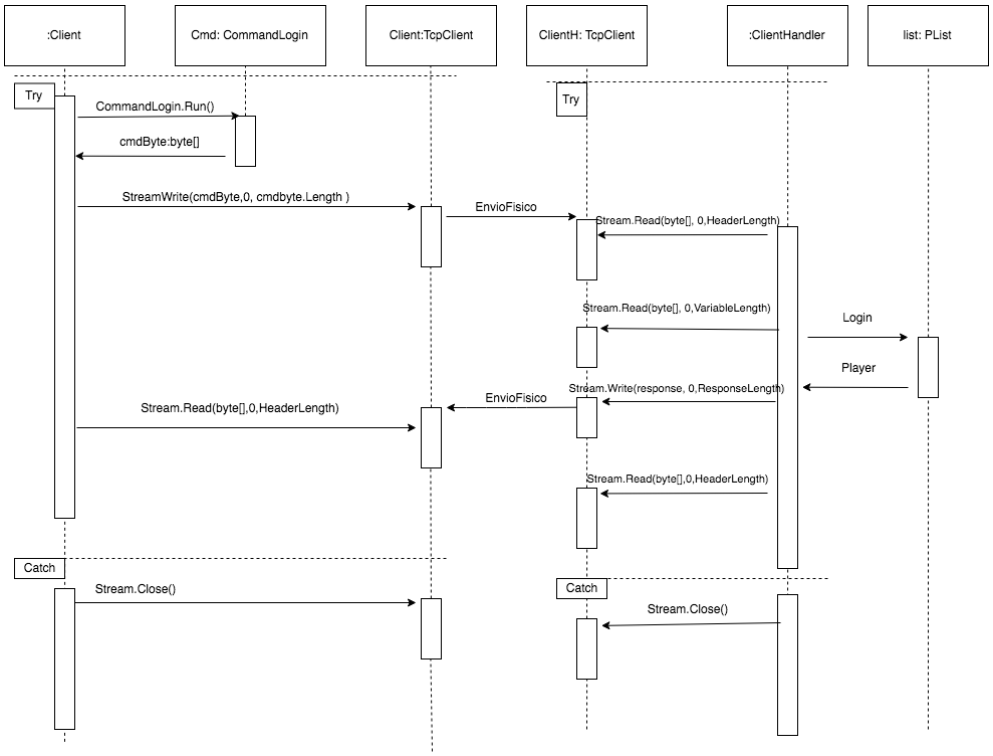
6.2.4 Proyecto GameServer

En las clases que heredan de Command se quiso poner que el método Run retorne byte[] pero como el StarUml no dejaba poner esa notación le pusimos bytearray.



6.3 Diagrama de Secuencia

En la siguiente imagen se puede ver el diagrama de secuencia de la funcionalidad del login que ejemplifica la comunicación tcp entre cliente y servidor.



En este diagrama se puede ver el catch de la excepción DisconnectedException usada para finalizar la conexión. Sin embargo, algunos detalles que no se reflejan y nos parecen importantes de aclarar son los siguientes. La conexión entre cliente y servidor fue previamente establecida. El cliente al ingresar comandos entra en el lock de Console.ReadLine y espera hasta que el cliente ingrese los datos para el comando. En este comando el cliente debe ingresar el usuario a loggear. El TcpClient del cliente es manejado por dos hilos en paralelo para los Read y Write asi pudiendo recibir datos del servidor mientras el otro hilo está siendo bloqueado por la consola. El login de player en PlayerList usa monitores para manejar la concurrencia de todos los ClientHandlers. La respuesta del servidor se imprime por consola. Y finalmente, el Stream.Read hecho al final por el Servidor es para mostrar como el servidor siempre está esperando mensajes del cliente.

7. Decisiones de diseño

- Todos los comandos del cliente requieren de una respuesta del servidor, aunque sea un simple "ok" antes de permitir al cliente seguir ingresando datos para poder manejar mejor los errores en caso de que se produzcan.

- Para atrapar cualquier tipo de perdida de conexión (sea voluntaria o involuntaria) se creó la excepcion DisconnectedException la cual es arrojada en caso de que se pierda la conexión. Esta es atrapada y lleva a que el que la atrapo se cierre además de que termine de ejecutar el hilo usado para procesar dichos requests.

- El juego se presenta todo por consola así permitiendo al desarrollo enfocarse más en la funcionalidad del sistema que en la estética, la cual consideramos poco relevante para la propuesta del obligatorio.

- Usamos el protocolo Tcp ya que creemos que este es más confiable para la transferencia de datos requerida por el programa solicitado. A su vez creemos que un envió de datos sincrónico facilita la concurrencia de pedidos al servidor hacienda que el Sistema no dependa tanto de monitores.

- Usamos TcpCients en lugar de Sockets ya que estos facilitan el envió de datos al no requerir que se cuente bit por bit la cantidad de datos haciendo su ejecución más confiable. Por los mismos motivos, en el servidor manejamos los pedidos de conexión con un TcpListener.

- Usamos monitores y únicamente los métodos "enter" y "exit" con un lock estático ya que la complejidad de concurrencia no exigía más que eso. Agregar complejidad a la concurrencia aumentaría el riesgo de bugs en el sistema.

- La creación de arrays de bytes a partir de los datos que ingresa el usuario se manejan en clases que heredan de la clase abstracta command. Esto se hizo para aumentar la cohesión de la solución, disminuir la cantidad de casos de "if else" del sistema y permitir una extensibilidad de comandos más sencilla. Cada comando simplemente ejecuta su método Run (), devuelve un byte [] y este es enviado por el stream del TcpClient.

- La clase server main no mantiene una lista de todos los tcpClients que crea y le asigna a cada handler para poder cerrar todas las conexiones en caso de que la ventana del servidor se cierre o que el usuario del servidor ingrese el comando "exit".

- Decidimos escribir todo el código en ingles ya que es un lenguaje universal.

8. Manual De Usuario

Se ejecuta el servidor y tantos clientes como se requiera.
Para esto, se debe configurar los parámetros “serverip”, “serverport” y “matchduration” en el archivo installConfig que se encuentra dentro del proyecto GameComm.
El serverip indica el IP de la máquina que se encuentra el servidor y matchduration se indica cuanto tiempo dura la partida en milisegundos.
Una vez que se configura esto y se le da en Iniciar, ya se puede empezar a ejecutar comandos (mencionados en los puntos 3.1 y 3.2) en la consola.

9. Bugs

Corrimos el programa varias veces y no encontramos ningún bug conocido, lo que no significa que no los haya.