Asier Santos y Endika Esteban

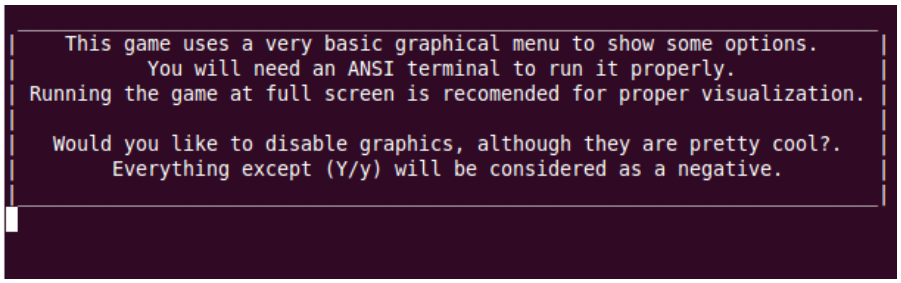
Practica general II

Servicios y aplicaciones en red

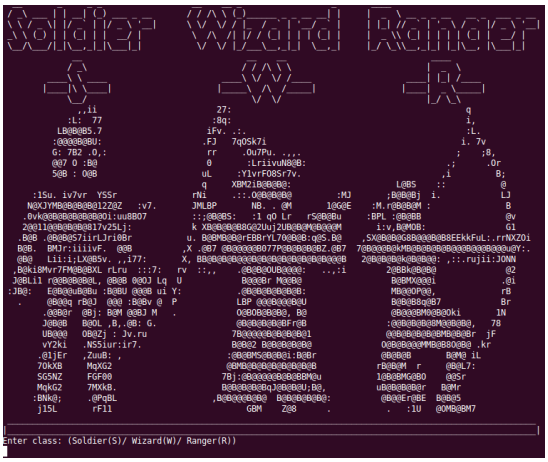
Introducción

Tal y como se explicó en la práctica general I la idea del proyecto es crear un juego de combate por turnos entre dos jugadores. La mecánica del juego consiste en tres clases y tres tipos de daño, el principal objetivo es reducir la vida del rival a 0 y para ello será necesario escoger adecuadamente el tipo de ataque que se emplea.

Antes de comenzar el protocolo se preguntara al jugador si desea emplear interfaz gráfica en la aplicación o no. El funcionamiento de la aplicación será independiente de esta opción, pues solo condiciona la forma en la que se presentaran los datos.



Cuando un usuario quiera jugar se identificará y decidirá el tipo de clase con la que jugara esa partida. Después de esto será puesto en una cola hasta que otro usuario se conecte y el juego comience.



El primero que se identificó comenzará atacando. Para atacar el usuario especificará el tipo de ataque que quiere hacer de entre los tres posibles y se calculará el daño en función de su clase, el tipo de ataque y la clase de su oponente (Ver nexo 1 – Calcular el daño).

Tras esto se informará al otro usuario de la vida que le queda tanto a él como a su oponente (ya que es el servidor que lleva la gestión de la vida y el cliente no lo almacena) y se le permitirá atacar. Este proceso continuará hasta que la vida de uno de los usuarios llegue a 0, teniendo una vida inicial de 100. En este punto se informará al vencedor de su victoria y al perdedor de su derrota y terminará la ejecución.

Protocolo modificado

Sintaxis y semántica

Finalmente se ha decidido que los comandos finalizarán con un carácter '#' y que los parámetros se separarán mediante el carácter '_'.

Protocolo

El protocolo de transporte que elegido para desarrollar la aplicación es UDP. Lo que nos interesa transmitir son mensajes breves, con poca información, por lo que no creemos necesario establecer una conexión constante entre cliente y servidor.

Comandos:

Comando	Parámetros	Descripción
USER	User_name (Char[])	Solicita inciar sesión con ese nombre de usuario.
ROLE	Class (Char)	Concretar la clase con la que se jugara la sesión. Class = S para Soldier, Class = W para Wizard y Class = R para Ranger
STRT	Initiative (Boolean)	Comando con el que el servidor notifica el inicio del combate. Initiative = True para el que juegue primero y Initiative = False para el que juegue segundo.
ATCK	Attack_type (Char)	Se emplea al realizar un ataque y se concreta el tipo de ataque que se realiza. Attack_type = M para Melee, Attack_type = S para Sorcery y Attack_type = R para ranged.
STAT	Your_hp (int) Enemy_hp (int)	El servidor notificara así al cliente el estado en el que se encuentra. Your_hp indica la vida restante al cliente que lo recibe, mientras que Enemy_hp indica la vida del rival.
VICT	Victory (Boolean)	Indica el final del combate. Si has vencido recibirás Victory = True, si no Victory = False.

Respuesta:

Comando	Parámetros	Descripción
OK		Respuesta positiva, todo ha ido bien.
ER	Err_Code (int)	Respuesta negativa ha habido algún problema.

Comandos compilación

Cliente

gcc cli_arena.c -o cli_arena

./cli_arena

Servidor

gcc ser_arena.c -o ser_arena

./ser_arena

Autoclient

gcc autoclient.c -o autocli

./autocli

Instrucciones de ejecución

1. Ejecutar el servidor
2. Ejecutar el primer cliente:
    I. Introducir el nombre de usuario.
    II. Introducir el rol.
3. Ejecutar el segundo cliente:
    I. Introducir el nombre de usuario.
    II. Introducir el rol.
4. Realizar el combate:
    I. A alternar ataques entre los clientes empezando por el primero.

Parámetros de los programas

Ni el cliente ni el servidor requieren de parámetro alguno a la hora de ser ejecutados.

Explicación del código

Cliente

La estructura de la función principal del cliente es:

1. Crear el socket.
2. Solicitar el nombre de usuario y enviarlo.
3. Solicitar el rol y enviarlo.
4. Esperar al emparejamiento.
5. Realizar el combate.

Explicaciones

En las funciones en las que se solicita información al usuario se efectúa el control de errores de forma que no se permita el envío de la misma hasta que los valores recibidos sean válidos (Tipo de rol, ataque...)

Al interpretar el estado del combate que se recibe se debe convertir el valor recibido en una string a un valor numérico con la vida que le queda a cada jugador. Para ello se emplea el siguiente código:

*MyHP=(buf[5]- '0')*100+(buf[6]- '0')*10+(buf[7]- '0')*

Al realizar la operación "- '0'" a un carácter obtenemos su valor numérico, los multiplicadores otorgan al dígito el valor real.

Servidor

La estructura de la función principal del servidor es:

1. Crear socket de identificación.
2. Bucle infinito:
    2.1. Esperar al jugador 1
    2.2. Esperar al jugador 2
    2.3. Crear proceso hijo que gestione el combate:
        2.3.1.Cerrar socket de identificación.
        2.3.2.Realizar combate
            2.3.2.1. Mandar iniciativa.
            2.3.2.2. Alternar ataques
            2.3.2.3. Mandar victoria

Explicaciones

El servidor usa dos sockets, uno para la identificación y otro para el combate. Además de esto se ha creado una variable (tempSocket) en la que se almacena el socket que se debe emplear en cada momento, de esta forma no son necesarias múltiples funciones para el envío y recepción de paquetes, simplemente se modifica socketTemp para que en cada situación se use el socket correcto.

El servidor gestiona el estado de cada uno de los jugadores y les envía a estos solo los datos que necesitan de los mismo, de forma que un cliente nunca será consciente de la dirección del otro cliente.

Decisiones implementación

Funciones de envío/recepción de datos agrupadas

Para no repetir el mismo código a lo largo de la implementación tanto de cliente como servidor hemos decidido dividir el envío de datos en dos tipos de funciones:

Funciones de preparación del comando

Por cada comando que pueda ser enviado se ha implementado una función de este tipo. La función obtiene los parámetros de las variables globales y carga en el buffer del programa una String con la información. Después de esto invoca la función de envío.

Función de envío

Esta función es única, su cometido es enviar los datos que ese momento se encuentran en el buffer al lugar al que se le indique.

Función de recepción

Esta función es única y se dedica a recibir la información y después de interpretarla colocarla en el buffer.

Interpretación de la información recibida

Para interpretar correctamente la información recibida se procesa la String recibida hasta encontrar el carácter de fin de comando y se sustituye por el de fin de String, de forma que se fija el tamaño de la misma.

Struct player

Para que el servidor almacene de forma sencilla los datos de cada jugador se ha creado esta struct:

```
struct player{
        char username[MAX_BUF];
        int role;
        int HP;
        struct sockaddr_in dir_cli;
};
```

Funciones que simulan comportamientos del sistema

Se ha desarrollado un auto-cliente que simula el comportamiento de un cliente conectándose al servidor. De esta forma es más fácil detectar fallos en la aplicación. El auto cliente simula de principio a fin todas las conexiones que un cliente debe realizar con el servidor, llevara a cabo todo el combate con un nombre parcialmente aleatorio y una clase y decisiones de ataque completamente aleatorias

Funcionalidades no implementadas

Cuentas de jugador

Para ser realmente funcional la aplicación requiere un control real sobre los usuarios y las sesiones. Podría almacenarse el nombre de usuario de forma que se pueda crear un perfil de jugador con una contraseña y se acceda al mismo siempre que se quiera jugar.

Post combate

La aplicación no contempla que sucede después del combate, pero sería recomendable añadir funcionalidades al final del combate tales como volver a jugar, volver a elegir la clase, etc.

Anexos

Calcular el daño

El daño que se hace al realizar un ataque depende de tres factores: La clase del atacante, el tipo de ataque que realiza y la clase del defensor.

Cada clase obtiene un daño cuando realiza un tipo de ataque

Ej: Un soldado atacando a melee hace mucho daño (x1,2), pero no es bueno atacando con magia por lo que no hará tanto daño (x0,8).

Atacante \ Tipo de Ataque	Melee (M)	Sorcery (S)	Ranged (R)
Soldier (S)	1.2	0.8	1
Wizard (W)	1	1.2	0.8
Ranger (R)	0.8	1	1.2

Cada clase padece de forma diferente un tipo de ataque

Ej: Un soldado que reciba un ataque de magia sufre mucho daño (x1,5) mientras que un ataque a distancia a penas le afecta (x0,5).

Tipo de Ataque \ Defensor	Soldier (S)	Wizard (W)	Ranger (R)
Melee (M)	1	1.5	0.5
Sorcery (S)	0.5	1	1.5
Ranged (R)	1.5	0.5	1

El daño total del ataque se calcula empleando la siguiente formula:
Daño total = 10 * (Modificador por atacante + modificación por defensor)

Ejemplo:

Un mago está combatiendo contra un explorador. Es el turno del mago, y decide atacar con magia.
El mago por atacar con magia obtiene un bono de 1,2, y un explorador resulta especialmente débil al sufrir un ataque mágico por lo que el ataque tiene un bono de 1,5.
*Daño total = 10 * (1,2+ 1,5) = 27*