PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2333 — Sistemas Operativos y Redes — 2/2021

Proyecto 2

29 de octubre, 2021

Fecha de Entrega: Lunes 15 de noviembre, 2021 hasta las 20:59
Fecha de ayudantía: Vídeo subido durante el fin de semana
Composición: grupos de 5 aldeanos (alumnos)

Índice

1. Objetivos

Este proyecto requiere que como grupo implementen un protocolo de comunicación entre un servidor y múltiples clientes para coordinar un juego multijugador. El proyecto debe ser programado en el **lenguaje C**. La comunicación entre las partes debe hacerse través de la funcionalidad de *sockets* de la API **POSIX**.

Requisitos fundamentales:

1. Los clientes de este juego deberán comportarse como *dumb-clients*. Esto significa que actúan únicamente de intermediarios entre el usuario y el servidor. Este último es quien se encargará de computar **toda** la lógica.

2. Debe existir obligatoriamente una interfaz por consola. En otras palabras, todos los efectos de las funcionalidades del juego deben verse reflejados en ella.

Nota: No se asignará el puntaje correspondiente para cada funcionalidad que no cumpla con estas restricciones.

2. Descripción: *IkaRuz*

El profesor Ruz lloró, pues no habían más tierras que conquistar. Como está ocupado manejando sus nuevas tierras le pidió a sus alumnos expertos en redes que creen el juego **IkaRuz** y permitirle al profesor conquistar las tierras virtuales de sus ayudantes.

El juego consiste en un juego de estrategia y manejo de recursos por turnos. En este juego participan 4 jugadores cada jugador tiene una aldea y una cierta población. Por turnos, cada jugador deberá tomar decisiones de donde destinar sus aldeanos para obtener recursos, pelear y crecer su aldea.

2.1. Conexión inicial e inicio del juego (10 puntos)

- **Conexión inicial** - El servidor en ejecución deberá esperar a que los clientes o jugadores se conecten para empezar el juego. El servidor debe estar preparado para mantener hasta 4 jugadores conectados de forma simultánea. Cuando un cliente se conecte al servidor se le debe dar la bienvenida. El primer jugador en conectarse será designado como el **líder del grupo**.
- **Lobby** - Al conectarse, todos los jugadores ingresan a un lobby inicial en donde se les preguntará por su **nombre** y se le pedirá que reparta 9 aldeanos en uno de los cuatro roles. El **líder del grupo** debe ser notificado sobre los nuevos jugadores que ingresen mostrando su nombre.
- **Inicio del juego** - El **líder del grupo** será el único con la opción de iniciar el juego. Si él es el único jugador conectado, o bien, si alguno de los jugadores conectados todavía no ha ingresado su nombre o no ha repartido sus aldeanos, no se podrá iniciar el juego, esto último no ocurrirá y se le notificará al **líder del grupo** que los jugadores no están listos.

2.2. Flujo del programa (5 puntos)

El flujo del programa será por **turnos**. El turno de cada jugador sigue el siguiente orden:

1. Recolectar recursos
2. Realizar acciones
3. Pasar de turno

2.3. Recolectar recursos (4 puntos)

Para poder crear soldados y crecer la aldea, el jugador deberá conseguir y gastar recursos. Los recursos disponibles son: oro, comida y ciencia. Al inicio de cada turno, el jugador recibirá una cantidad de estos recursos que dependerá de la cantidad de aldeanos que tiene asignados a extraer esos recursos multiplicado por su nivel producción.

Al inicio de cada turno, cada jugador recibe las siguientes cantidades de cada recurso:

- **Oro**: $cantidad\_de\_mineros \times nivel\_mineros \times 2$
- **Comida**: $cantidad\_de\_agricultores \times nivel\_agricultores \times 2$
- **Ciencia**: $cantidad\_de\_ingenieros \times nivel\_ingenieros$

2.4. Roles (4 puntos)

Durante su turno, cada jugador podrá gastar sus recursos para crear nuevos aldeanos. Cada aldeano creado debe tener un rol. El costo de producir cada aldeano y sus características son las siguientes:

2.4.1. Agricultores

Los agricultores son los responsables de obtener el recurso **comida**, entre más agricultores más **comida** generarás al inicio del turno. El costo de creación es 10 de **comida**.

2.4.2. Mineros

Los mineros son los encargados de obtener el recurso **oro**. Su costo de creación es 10 de **comida** y 5 y **oro**.

2.4.3. Ingenieros

Los ingenieros son los encargados de aumentar la producción de **ciencia** de la aldea. Su costo de creación es 20 de comida y 10 de oro.

2.4.4. Guerreros

Los guerreros tienen el rol de **defender** la aldea o **atacar** otras aldeas. El costo de creación es de 10 de **comida** y 10 de **oro**.

2.5. Realizar acciones (20 puntos)

Luego de obtener recursos, un jugador podrá realizar tantas acciones como quiera hasta pasar de turno. Las acciones posibles son las siguientes:

2.5.1. Mostrar información (2 puntos)

Cuando un usuario hace esta acción se le deberá mostrar en pantalla toda la información necesaria para administrar su aldea. Como mínimo, está acción deberá imprimir la siguiente información:

- cantidad de cada recurso
- cantidad de aldeanos en cada rol
- nivel de agricultores, mineros, ingenieros, ataque y defensa

2.5.2. Crear aldeano (2 puntos)

Con esta acción el usuario puede gastar su **comida** y **oro** para crear nuevos aldeanos. El costo de creación depende del rol que se le quiera asignar al aldeano.

2.5.3. Subir de nivel (2 puntos)

Con la **ciencia** producida por los ingenieros se puede mejorar la infraestructura de la aldea, las herramientas y armas de sus aldeanos. Todos los roles tienen asociados un nivel (los guerreros tienen asociado defensa y ataque) que puede ser subido con la **ciencia**.

Al hacer la acción **subir nivel**, el jugador debe elegir cual es el aspecto que se desea mejorar de la aldea, estos son:

- Nivel agricultores
- Nivel mineros
- Nivel ingenieros
- Nivel ataque
- Nivel defensa

Al inicio del juego, todos los niveles comienzan se inicializan en 1 y el costo para subir a cada nivel es el siguiente:

- Nivel 2: 10 de **comida** + 10 de **oro** + 10 de **ciencia**
- Nivel 3: 20 de **comida** + 20 de **oro** + 20 de **ciencia**
- Nivel 4: 30 de **comida** + 30 de **oro** + 30 de **ciencia**
- Nivel 5: 40 de **comida** + 40 de **oro** + 40 de **ciencia**

Los niveles se suben de manera secuencial y ningún nivel puede ser superior a 5. Si no se cuentan con los recursos necesarios, se debe mantener en el mismo nivel e informar al usuario; en caso contrario, los recursos deben reflejar que fueron utilizados.

2.5.4. Atacar (5 puntos)

Como una acción, un usuario puede atacar la aldea de otro usuario. Para hacer esto se compara la fuerza de su ejército versus la fuerza del ejército del usuario atacado. Si el usuario es quien ataca, su fuerza se calcula de la siguiente manera:

$$Fuerza = cantidad\_de\_guerreros \times nivel\_ataque$$

Si el usuario es quien defiende, su fuerza se calcula de la siguiente manera:

$$Fuerza = cantidad\_de\_guerreros \times nivel\_defensa \times 2$$

Si la fuerza del atacante es mayor que la fuerza del defensor, entonces el atacante gana toda la **comida**, **oro** y **ciencia** del defensor. Además, el defensor es eliminado de la partida. Un jugador eliminado no puede ser atacado y el servidor deberá "saltarse" su turno. En caso contrario, el atacante pierde la mitad de sus guerreros (redondeado hacia abajo). En cualquier caso, luego continua su turno normalmente.

2.5.5. Espiar (3 puntos)

Se podrá espiar a una aldea enemiga para conocer el estado de su ejercito (la cantidad de guerreros, nivel de ataque y defensa). El costo de espiar es de 30 de **oro**, y el gasto de estos debe verse reflejado apenas se comience con la acción.

2.5.6. Robar (3 puntos)

Como una acción, puedes gastar 10 de **Ciencia** para robarle recursos a otros jugadores. Esta acción te debe permitir seleccionar a otro usuario y elegir entre **Comida** y **Oro**. El usuario seleccionado deberá perder el 10 % del recurso seleccionado y el usuario que realizó la acción deberá ganar la misma cantidad del recurso.

2.5.7. Pasar (1 punto)

Termina el turno actual.

2.5.8. Rendirse (2 punto)

Los usuarios pueden rendirse en cualquier momento de la partida. Una vez se rinda, todos lo recursos y aldeanos de la aldea desaparecen. Queda a su criterio si el usuario queda como espectador o se desconecta.

2.6. Interfaz por consola (7 puntos)

Se espera que para este proyecto desarrollen una **interfaz por consola** para que los jugadores puedan interactuar con el juego y ser informados de lo que ocurre. La interfaz debe ser ordenada y clara.
En cada turno se deben recibir los recursos y presentar el menú de acciones debe aparecer en pantalla. Al elegir una opción que involucre una opción secundaria, deben mostrar otro menú con las opciones (como qué tipo de aldeano crear, o qué aldea atacar).
Fuera del turno se deben imprimir los mensajes que informen sobre el desarrollo del juego. Esto incluye informar a todos los jugadores acerca de ataques, jugadores que se rinden, robos y cuando un jugador pierde. En estos mensajes se debe detallar los jugadores que participan de estas acciones. En el caso de espionaje, no se debe notificar nada. Queda a su criterio como imprimir la información pero deben cumplir con presentarle todas las acciones y notificarlo de los eventos importantes.

3. Implementación

3.1. Cliente

El cliente debe recibir e imprimir los menús y/o mensajes correspondientes y debe enviar los inputs al servidor.

3.2. Servidor

El servidor debe mediar la comunicación entre los clientes. El servidor es el encargado de procesar **toda** la lógica del juego (recibir y procesar *inputs* de usuario, mantener el estado de cada participante de un combate, gestionar cada turno, entregar recursos, manejar ataques, robos y espionajes, etc.).

3.3. Protocolo de comunicación (7 puntos)

Todos los mensajes enviados, tanto de parte del servidor, como de parte del cliente, deberán seguir el siguiente formato:

- `ID` (1 *byte*): Indica el tipo de paquete.
- `PayloadSize` (1 *byte*): Corresponde al tamaño en *bytes* del `Payload` (entre 0 y 255).
- `Payload` (`PayloadSize` *bytes*): Es el mensaje propiamente tal. En caso de que no se requiera, el `PayloadSize` será `0` y el `Payload` estará vacío.

Tienen total libertad para implementar los paquetes que estimen necesarios. No obstante, deberán documentarlos todos en su `README.md`, explicitando el `ID` y el formato de `Payload`, junto con una breve descripción de cada uno.

A modo de ejemplo, consideren que queremos enviar el mensaje `Hola` con el `ID` 5. Si serializamos el `Payload` según **ASCII**, su tamaño correspondería a cuatro *bytes*. El paquete completo se vería así:

```
00000101 00000100 01001000 01101111 01101100 01100001
   5        4         H        o        l        a
```

3.4. Ejecución (3 puntos)

Los clientes y el servidor deberán ejecutarse de la siguiente manera, respectivamente:

```
$ ./server -i <ip_address> -p <tcp_port>
$ ./client -i <ip_address> -p <tcp_port>
```

Donde `<ip_address>` corresponde a la **dirección IP** del servidor (en formato numérico de IPv4, por ejemplo, `172.16.254.1`) y `<tcp_port>` al puerto TCP a través del cual el servidor recibirá nuevas conexiones.

El proyecto solo será corregido si cumple con esta modalidad de ejecución.

4. Bonus (¡hasta 15 décimas!)

4.1. *SSH Tunneling* (+5 décimas)

Su programa debe poder funcionar de forma completamente distribuida. Esto es, con el servidor ejecutando en `iic2333.ing.puc.cl`, y clientes ejecutándose en lugares distintos (como sus domicilios, por ejemplo). El servidor `iic2333.ing.puc.cl` posee abierto solamente los puertos 22 y 80.

Este desafío deberá ser resuelto haciendo uso de **SSH Tunneling** para poder conectarse con un puerto local de `iic2333.ing.puc.cl`, un mecanismo cuyo funcionamiento deberán investigar por cuenta propia. Se recomienda revisar **este link** como punto de partida.

4.2. *Negociar* (+10 décimas)

Dentro de las acciones se deberá poder hacer negociaciones de recursos entre usuarios. Un usuario puede iniciar una negociación en su turno indicando que recurso y cuanto esta dispuesto a entregar a cambio de otro recurso y su cantidad respectiva. Esta negociación debe ir dirigida a otro usuario y es este último quién decide si aceptar, rechazar o modificar la propuesta. En caso de ser aceptada, la cantidad de recursos de cada jugador deben reflejar el intercambio. En caso de ser rechazada, se debe informa a todos los usuarios que hubo una negociación fallida. Por último, si la propuesta es modificada, se envía al otro usuario. Esto se puede repetir hasta que se acepte o rechace la negociación.

5. *README* y formalidades

5.1. Grupos

Está permitido que los grupos cambien respecto al proyecto anterior.

5.2. Entrega

Su proyecto debe encontrarse en un directorio con nombre P2 (mayúscula la P) dentro del directorio de UNO de los integrantes del grupo en el servidor del curso. Además, deberán incluir:

- Un archivo `README.md` que indique quiénes son los autores del proyecto (**con sus respectivos números de alumno**), instrucciones para ejecutar y usar el programa, descripción de los paquetes utilizados en la comunicación entre cliente y servidor, cuáles fueron las principales decisiones de diseño para construir el programa, cuáles son las principales funciones del programa, qué supuestos adicionales ocuparon, y cualquier información que consideren necesaria para facilitar la corrección de su tarea. Se recomienda usar formato **Markdown**.
- Uno o dos archivos `Makefile` que compilen su programa en dos ejecutables llamados `server` y `client`, correspondientes al servidor y al cliente, respectivamente.

5.3. Otras consideraciones

- Este proyecto **debe** ser programado usando el lenguaje C. No se aceptarán desarrollos en otros lenguajes de programación.
- No respetar las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos.

6. Descuentos

6.1. Descuentos (¡hasta 20 décimas!)

- 5 décimas por subir archivos binarios (programas compilados).
- 5 décimas por no incluir el/los `Makefiles`, o bien incluirlos y que no funcionen.
- 5 décimas por tener *memory leaks*. (Se recomienda fuertemente utilizar **Valgrind**).
- 5 décimas por la presencia archivos correspondientes a entregas del curso pasadas en el mismo directorio.

6.2. SÚPER DESCUENTO (¡La mitad de la nota!)

Si por cualquier motivo su proyecto no funciona ocupando sockets esto generará que cualquier puntaje que se haya obtenido se reduzca a la mitad. (¡Como hay puntaje por la conexión inicial esto significa que el 4 es imposible!)