



IIC2333 — Sistemas Operativos y Redes — 1/2020  
**Proyecto 2**

Lunes 08-Junio-2020

**Fecha de Entrega: Lunes 22-Junio-2020**

**Composición: grupos de  $n$  personas, donde  $3 \leq n \leq 4$**

**Fecha de ayudantía: Viernes 12-Junio-2020**

## Índice

<b>1. Objetivos</b>	<b>2</b>
<b>2. Descripción: <i>Super Ruz Party</i></b>	<b>2</b>
2.1. Inicio del juego (10 puntos)	2
2.2. Movimiento de los jugadores (4 puntos)	3
2.3. Recursos	3
2.4. Poderes (5 puntos)	3
2.5. Interfaz por consola	3
2.6. Casillas del tablero (17 puntos)	4
2.7. Turnos (10 puntos)	5
2.8. Duelos (7 puntos)	5
<b>3. Implementación</b>	<b>6</b>
3.1. Cliente	6
3.2. Servidor	6
3.3. Protocolo de comunicación (7 puntos)	6
3.4. Ejecución	6
<b>4. Bonus (¡hasta 15 décimas!)</b>	<b>7</b>
4.1. Envío de Imágenes (+5 décimas)	7
4.2. Juegos simultáneos (+5 décimas)	7
4.3. <i>SSH Tunneling</i> (+5 décimas)	8
<b>5. <i>README</i> y formalidades</b>	<b>8</b>
5.1. Grupos	8
5.2. Entrega	8
5.3. Otras consideraciones	8
<b>6. Descuentos (¡hasta 20 décimas!)</b>	<b>8</b>
<b>7. Nota final y atraso</b>	<b>9</b>

# 1. Objetivos

Este proyecto requiere que como grupo implementen un protocolo de comunicación entre un servidor y dos clientes para coordinar un juego en línea. 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. Al finalizar el juego, debe aparecer un mensaje indicando la razón por la cual finalizó y se debe mostrar claramente al ganador.
3. 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 tres restricciones.

## 2. Descripción: *Super Ruz Party*

El profesor Cristian Ruz, preso por la nostalgia de poder hacer clases presenciales y de poder ver y conversar en persona con sus colegas y alumnos del DCC, se le ha ocurrido la genial idea de hacer un juego que emule dichos recuerdos: *Super Ruz Party*. Este es un juego de mesa en línea para dos jugadores, en el que cada uno es un alumno de computación que se esfuerza para cumplir sus tareas y proyectos. Para esto, el profesor les ha confiado a ustedes, sus esforzados alumnos, la tarea de programarlo.

### 2.1. Inicio del juego (10 puntos)

El juego consiste en una competencia entre dos jugadores. El primer cliente en conectarse con el servidor será el jugador 1 y el segundo será el jugador 2. **(2 puntos)**


Al momento de conectarse, se le deberá solicitar el nombre al usuario recién conectado. Este nombre será almacenado por el servidor y se utilizará por el resto del juego. Luego, se le deberá mostrar al jugador un mensaje de bienvenida personalizado (es decir, que contenga el nombre) generado por el servidor (*e.g.*, ¡Bienvenido, John Doe!). **(3 puntos)**

Luego de que ambos jugadores se hayan conectado exitosamente, hayan ingresado sus nombres y hayan recibido sus mensajes de bienvenida, el jugador 1 deberá ingresar la ruta de un archivo `.txt` con la información del tablero, la que debe ser usada por su servidor para generar el tablero del juego. **(3 puntos)**

Este archivo contiene 2 líneas:

- Primera línea: Un `int` que indica el largo del tablero, es decir, el número de casillas.
- Segunda línea: Un `string` de largo igual al `int` anterior en donde cada carácter representa una casilla del tablero. Las casillas del tablero se explican **más adelante**.

A modo de ejemplo, consideren el siguiente archivo:

 tablero1.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

40

I00P00X0Y0D0X00XX00T00X00XP000XAX00X0FRE

Una vez cargado el tablero comienza el juego. El primer turno corresponde al jugador 1. **(2 puntos)**

## 2.2. Movimiento de los jugadores (4 puntos)

Los jugadores se mueven a lo largo de las casillas un tablero unidimensional (1D) a través de una de las siguientes configuraciones:

1. **Modalidad aleatoria.** El jugador lanza un dado virtual de 10 caras equiprobables (los números van del 1 al 10, y tienen la misma probabilidad de ocurrencia). El jugador avanza hacia la derecha el número resultante de este lanzamiento. **(2 puntos)**
2. **Modalidad manual.** En este caso, el jugador hace “trampa”. En vez de lanzar el dado, elige un número del 1 al 10 (inclusive) de casillas que quiere avanzar hacia la derecha. **(2 puntos)**

## 2.3. Recursos

Cada jugador cuenta con distintos recursos:

- **Méritos:** Son el recurso principal del juego y se usan para obtener Estrellas Ruz y comprar poderes. La cantidad de méritos de cada jugador **nunca** puede ser menor que 0.
- **Estrellas Ruz:** Son el objetivo del juego. El primer jugador en conseguir 3 será el ganador.
- **Poderes:** Un jugador puede elegir activar uno al inicio del turno. Tienen distintos efectos. Un jugador puede tener hasta 3 poderes en todo momento.

## 2.4. Poderes (5 puntos)

Los poderes y sus efectos se detallan a continuación:

- **Dado Doble:** Al activarlo el jugador lanza 2 dados para moverse el número de casillas correspondiente a la suma de esos dos lanzamientos. **(3 puntos)**
- **Duelo:** Al activarlo el jugador reta al otro jugador a un duelo.
- **Roba Méritos:** Al activarlo el jugador le roba méritos al otro jugador. La cantidad de méritos robada es un número aleatorio entre 1 y 10 incluidos. La cantidad de méritos de cada jugador **nunca** puede ser menor que 0. **(2 puntos)**

## 2.5. Interfaz por consola

Un tablero de ejemplo junto con la información de los jugadores se ve de la siguiente manera:

1
2
  
I O O P O O X O Y O D O X O O X X O O T O O X O O X P O O O X A X O O X O F R E

---

Jugador 1:

- Nombre: Mario
- Méritos: 21
- Estrellas Ruz: 1
- Poderes:
  - Roba Méritos
  - Dado Doble
  - Roba Méritos

Jugador 2:

- Nombre: Luigi
- Méritos: 3
- Estrellas Ruz: 2
- Poderes:
  - Duelo
  - Duelo
  -

En esta imagen, 1 y 2 representan la posición actual de los jugadores 1 y 2 en el tablero, respectivamente, y las letras representan el tipo de casilla (cuyo significado será detallado [más adelante](#)).

**Nota:** Esta es solo una referencia. Ustedes son libres de decidir cómo van a mostrar el tablero y la información de cada jugador, siempre y cuando sea de forma clara.

## 2.6. Casillas del tablero (17 puntos)

El tablero del juego debe ser generado a partir de un archivo `.txt`. Este es ingresado por el jugador 1 antes de comenzar el juego. Este archivo contiene 2 líneas: la primera es un `int` que indica el largo del tablero o número de casillas del tablero, la segunda es un `string` en el que cada carácter representa una casilla. Las casillas junto a los caracteres que las representan se detallan a continuación:

- **I - Inicio:** Esta es la casilla en la que comienzan ambos jugadores al inicio del juego. Solamente existe una casilla I y esta debe ser la primera del tablero. **(1 punto)**
- **O - Méritos:** Un jugador que termina su turno en esta casilla gana 5 méritos. **(1 punto)**
- **X - Distracción:** Un jugador que termina su turno en esta casilla pierde 5 méritos. La cantidad de méritos de cada jugador **nunca** puede ser menor que 0. **(1 punto)**
- **R - Cristian Ruz:** Un jugador que termina su turno en esta casilla conseguirá el aprecio del profesor (1 Estrella Ruz) a cambio de 20 méritos. Esto sucederá de forma automática en caso de que el jugador tenga los 20 méritos. En caso contrario, esta casilla no tendrá ningún efecto. Si un jugador consigue su tercera Estrella Ruz, es declarado como ganador y termina el juego. **(3 puntos)**
- **P - Poder:** Un jugador que termina su turno en esta casilla gana un poder al azar. Un jugador no puede tener más de 3 poderes al mismo tiempo, por lo que esta casilla solo tendrá efecto solo si el jugador posee menos de 3 poderes. **(2 puntos)**
- **T - Tienda:** Un jugador que termina su turno en esta casilla puede comprar un poder a un precio de 10 méritos. Si el jugador ya tiene 3 poderes, esta casilla no tendrá ningún efecto. **(2 puntos)**
- **F - Súper F:** Un jugador que termina su turno en esta casilla vuelve inmediatamente a la casilla de inicio (I). **(1 punto)**
- **E - Salida:** Un jugador que pase por esta casilla vuelve al inicio (I) y gana 20 méritos. El tablero es cíclico, por lo que, al volver a la casilla de inicio, el jugador continuará avanzando si le corresponde. **(2 puntos)**

- A - **Sala Álvaro Campos:** Un jugador que termina su turno en esta casilla trabaja arduamente en cumplir con sus tareas y proyectos y gana 30 méritos. **(1 punto)**
- D - **Duelo:** Un jugador que termina su turno en esta casilla reta al otro jugador a un duelo. Los detalles del duelo se explican **más adelante**.
- Y - **Fonda Don Yadrán:** Un jugador que termina su turno en esta casilla pierde su próximo turno (en otras palabras, su contrincante jugará dos veces seguidas). **(3 puntos)**

## 2.7. Turnos (10 puntos)

El juego es por turnos. Esto significa que solo hay un jugador activo a la vez. Mientras un jugador esté realizando su turno, el otro no debe hacer más que esperar. Esto quiere decir que el jugador inactivo (*i.e.*, el que está esperando) queda imposibilitado de ejecutar cualquier acción y en caso de intentar interactuar con la consola deberá ser ignorado. **(2 puntos)**

Por otro lado, el turno del jugador activo se desarrolla de la siguiente manera y en este orden:

1. Se muestra en pantalla el tablero actualizado (incluyendo los méritos, Estrellas Ruz y poderes de ambos jugadores) antes de avanzar. Es decir, como quedó en el turno pasado. **(2 puntos)**
2. Se muestra en pantalla un menú con todas las opciones habilitadas (solo puede elegir una):
  - Utilizar un poder. Los poderes se explicaron **acá**. Un jugador solo puede usar un solo poder en un mismo turno.
  - Rendirse. El jugador puede rendirse. Si decide hacerlo, se debe mostrar en ambas pantallas que el ganador es el contrincante y luego finalizar el juego. **(2 puntos)**
  - Avanzar. El jugador elige una modalidad (aleatoria o manual) y avanza el número de casillas correspondiente. Estas modalidades se explicaron **acá**.
3. La casilla en donde termina el jugador después de moverse se activa. Las casillas se explicaron **acá**.
4. Se muestra en pantalla el tablero actualizado (incluyendo los méritos, Estrellas Ruz y poderes de ambos jugadores) después de finalizar el turno. **(2 puntos)**
5. Si el juego no ha terminado, comienza el siguiente turno. **(2 puntos)**

## 2.8. Duelos (7 puntos)

Un duelo corresponde a un juego de *cachipún*, que se desarrolla de la siguiente manera y en este orden:

1. Este puede iniciarse de tres maneras: (1) **terminando** el turno en una casilla D, (2) a través del Poder Duelo y (3) cuando un jugador **termina** en la misma casilla que su contrincante. **(3 puntos)**
2. Si sucede que un jugador termina en una casilla D que es, a la vez, donde se encuentra su contrincante, se deberán realizar dos duelos seguidos. **(2 puntos)**
3. Cada jugador elige entre *piedra*, *papel* y *tijera*. **(1 punto)**
4. El ganador del duelo le quita 10 méritos al perdedor. En el caso de empate, ambos jugadores ganan 5 méritos. **(1 punto)**

## 3. Implementación

### 3.1. Cliente

El cliente debe tener algún tipo de interfaz en consola que permita visualizar el tablero, la información de cada jugador, informar sobre efectos de casillas y poderes, y enviar *inputs* de usuario. No es necesario que todo se muestre al mismo tiempo, pero se espera que diseñen una interfaz tipo menú navegable que permita realizar las funcionalidades mencionadas, así como respetar los turnos de los jugadores. En este sentido, siéntase libres de diseñar esto como quieran, siempre que se entienda el flujo del juego y se cumplan con los requisitos mínimos de este.

Otro punto importante, que fue mencionado anteriormente, es que el cliente debe comportarse como un *dumb-client*, es decir, que solamente actúa de intermediario entre el usuario y el servidor y no es un participante activo de la lógica del programa.

### 3.2. Servidor

El servidor que ustedes implementarán 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 jugador, gestionar y comunicar el sistema de turnos entre los jugadores, evaluar quien es el ganador, 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. **(1 puntos)**
- PayloadSize (1 *byte*): Corresponde al tamaño en *bytes* del Payload (entre 0 y 255). **(2 puntos)**
- Payload (PayloadSize *bytes*): Es el mensaje propiamente tal. En caso de que no se requiera, el PayloadSize será 0 y el Payload estará vacío. **(2 puntos)**

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. **(2 puntos)**

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

```
00000101 00000100 01001000 01101111 01101100 01100001
```

Si necesitan inspiración, los invitamos a revisar los enunciados de semestres anteriores, en los que encontrarán algunos ejemplos de paquetes que podrían llegar a ser útiles (luego de algunas modificaciones) en este proyecto.

### 3.4. Ejecución

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 *human-readable*, 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!)

Para que esta sección sea tomada en consideración, es importante que indiquen en su `README.md` cuáles son los bonus implementaron y una breve descripción de su funcionamiento.

### 4.1. Envío de Imágenes (+5 décimas)

Se define un paquete especial para enviar imágenes desde el servidor a los clientes llamado *Image*. Como los paquetes de envío de datos entre el servidor y los clientes tienen un `Payload` de un tamaño máximo de 255 bytes, una imagen deberá ser enviada a través de varios segmentos. La estructura de este nuevo paquete es el siguiente:

- `ID` (1 byte): Indica el tipo de paquete.
- `TotalPackages` (1 byte): Corresponde al número total de paquetes necesarios para enviar la imagen.
- `CurrentPackage` (1 byte): Es el índice del paquete que se está enviando (se parte desde 1).
- `PayloadSize` (1 byte): Corresponde al tamaño en bytes del `Payload` que se está enviando (entre 0 y 255).
- `Payload` (`PayloadSize` bytes): Es la información de la imagen propiamente tal.

Por ejemplo, supongamos que la imagen que quiero enviar pesa 2552 bytes. Corresponde enviar un total de 11 paquetes ( $255 \cdot 10 + 2 \cdot 1$ ). Además, supongamos que los últimos 2 bytes son `0x0A` y `0x9F`. Si el `ID` del envío de imágenes es 64, entonces el último paquete en enviarse se vería así:

```
1000000 00001011 00001011 00000010 00001010 10011111
```

El cliente, al recibir el primer paquete de este tipo, tendrá que almacenar su información en un *buffer* y esperar a que el servidor le envíe todos los *payloads* pendientes. Que el `TotalPackages` sea igual al `CurrentPackage` significa que el servidor ha terminado de enviarle los datos de la imagen. Por lo tanto, el cliente va a poder guardar toda la información recibida en un solo archivo.

Las imágenes se almacenarán en el directorio `images/` en la raíz del servidor. Los nombres de las imágenes serán `winner.png` y `loser.png` y deberán ser enviadas correctamente al cliente ganador y perdedor, respectivamente, una vez terminado el juego.

### 4.2. Juegos simultáneos (+5 décimas)

El servidor estará preparado para aceptar conexiones de más de 2 clientes a la vez y será capaz de mantener más de una partida de *Super Ruz Party* al mismo tiempo. Para ordenar las partidas se deberá implementar un *lobby* con salas a las que un usuario podrá acceder. Las salas son creadas por los mismos usuarios y deben estar visibles para todos. Se deberá implementar este *lobby* con un máximo de 5 salas de 2 jugadores cada una (por tanto, 10 clientes conectados al mismo servidor).

Para esto, cuando un cliente se conecte al servidor, se le dará la bienvenida y se le mostrará en pantalla el menú del *lobby*, con una lista de salas (y la cantidad de jugadores en cada una), la opción de unirse a las salas en caso de que estás todavía no estén llenas y, por último, la posibilidad de crear una nueva sala si hay menos de 5.

El usuario que crea una sala debe asignarle un nombre y, una vez creada, ingresa automáticamente y espera a que se conecte un segundo usuario para comenzar el juego. Si una sala queda sin jugadores, es eliminada automáticamente.

Al terminar un juego, los clientes pueden volver a jugar, o bien salir de la sala. Si ambos deciden jugar de nuevo, el juego comienza otra vez. Si solo uno decide jugar de nuevo, espera a que otro se conecte. Al salir de la sala, se vuelve al *lobby*.

### 4.3. *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 dos clientes ejecutando 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.

## 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 la *master branch* de su repositorio en Github al momento de la entrega. Si realizan el **bonus 3**, también deberán subir su proyecto al servidor del curso (`iic2333.ing.puc.cl`).

Además, deberán incluir:

- Un archivo `README.md` que indique quiénes son los autores del proyecto (**con sus respectivos números de alumno**), instrucciones para ejecutar 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](#).
- Un o dos archivos `Makefiles` 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.
- Si bien no será evaluado, se recomienda el uso de [Valgrind](#).

## 6. Descuentos (¡hasta 20 décimas!)

- 5 décimas por subir archivos binarios (programas compilados).
- 10 décimas por no incluir el/los `Makefiles`, o bien incluirlos y que no funcionen.
- 5 décimas por la presencia archivos correspondientes a entregas del curso pasadas en la misma *branch*.



## 7. Nota final y atraso

La nota final del proyecto entregado a tiempo se calcula de la siguiente manera:

$$N = 1 + \frac{\sum_i p_i}{10} + b - d$$

Se puede hacer entrega del proyecto con un máximo de cuatro días de atraso. La fórmula a seguir es la siguiente:

$$N_{atraso} = \min(N; 7 + b - d - 0,75 \cdot a)$$

Siendo  $d$  el descuento total,  $p_i$  el puntaje obtenido en el ítem  $i$ ,  $b$  el bonus total,  $a$  la cantidad de días de atraso y  $\min(x; y)$  la función que retorna el valor mas pequeño entre  $x$  e  $y$ .