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

IIC2233 - Programación Avanzada (II/2016)
Tarea 6

1. Objetivos

- Aplicar conceptos y nociones de networking para la creación de un servicio en linea.
- Aplicar conceptos del serialización para el manejo de envio de archivos.
- Aplicar conocimientos de interfaces.

2. Introducción

Luego de pasar años en tu tanque luchando contra las maquinas y no lograr ganar, solo te queda resignarte y comenzar a vivir en tu tanque. Afortunadamente encuentras dentro de este vehículo un sistema de comunicación basado en envío de bytes y, sin nada mas que hacer, decides mejorar el sistema con la esperanza de lograr comunicarte con otras personas (si es que hay alguien mas vivo). Te das cuenta que con tus grandes habilidades de programación, la comunicación mas avanzada que puedes gestionar es mediante chat y dibujos. Es así como decides crear tu propio programa capaz de mandar dibujos en tiempo real y chatear, como eso sonaba aburrido y tu todavía no perdías la esperanza de que otra persona se comunicara contigo decides crear todo en forma de juego.

3. Programillo

La tarea consiste en implementar **Programillo**, una aplicación que permite a personas conectarse mediante una red y poder jugar un juego que consiste en adivinar una palabra a través de un dibujo hecho en la interfaz donde gana el jugador o el equipo que más palabras o frases adivine. También se deberá implementar una red social personal para la comunidad del juego, aplicando conceptos de networking y de serialización. Para esto usted deberá crear dos cosas: un servidor que correrá en su computador y manejará la lógica del sistema y un cliente que podrá correr en cualquier computador que permitirá la interacción de un usuario con el sistema. En los siguientes puntos se explicará en detalle todos los requerimientos que **Programillo** debe implementar.

4. Cliente-Servidor

Su implementación de Programillo deberá basarse en la arquitectura *cliente-servidor*, en donde todas las interacciones que se quieran realizar con algún usuario deberán pasar por el servidor. Para lograr lo anterior, deberá implementar un protocolo eficiente de comunicación entre el cliente y el servidor usando el modelo **TCP/IP**.

Todas las interacciones que se esperen del cliente deben realizarse a través de interfaces gráficas. La interfaz debe ser amigable y enfocada en la usabilidad, es decir, comportarse de acuerdo a lo que espera el usuario de forma natural, sin eventos inesperados.

4.1. Sistema de autentificación (registro e ingreso de usuarios)

Al iniciar el programa, este deberá ofrecer dos opciones: registrarse si es un usuario nuevo, o ingresar si es que es antiguo. Si es que se escoge la opción de registro, el chat deberá pedirle al usuario un nombre de usuario, contraseña y confirmación de la contraseña. Se debe verificar que el nombre de usuario no esté registrado, es decir, deben ser únicos entre todos los usuarios de Programillo.

La base de datos de los usuarios debe persistir. Dicho de otra forma, los usuarios registrados durante una ejecución del programa deberán estar disponibles durante una ejecución en un momento diferente. Para hacer esto queda a su criterio el mecanismo a implementar.

Existe un requerimiento especial de seguridad del sistema: las contraseñas nunca deben quedar guardadas en el disco duro sin encriptar. Para encriptar las contraseñas se debe usar el protocolo **hash + salt** que se detalla a continuación:

Cuando se escriba un nuevo usuario al disco duro debes:

1. Generar una secuencia de bytes aleatoria. Esta secuencia será nuestra 'sal'. Se recomienda fuertemente que la obtenga a través del método `urandom` del módulo `os`.
2. Concatenar la 'sal' con la codificación en bytes de la contraseña del nuevo usuario. Recuerda que puedes obtener la codificación en bytes a través del método `encode` de un string. La concatenación de ambas secuencias será nuestra secuencia a encriptar.
3. Pasar la secuencia a encriptar por alguna función de hash. Se recomienda fuertemente usar el algoritmo `sha256` a través del módulo `hashlib`. La secuencia de bytes que sea retornada por la función de hash será nuestra contraseña encriptada.

Luego, cuando un usuario intente ingresar a Programillo debes realizar lo siguiente:

1. Obtener desde la base de datos de Programillo la 'sal' y contraseña encriptada del usuario que está intentando ingresar.

2. Concatenar la sal con la secuencia de bytes de la codificación de la contraseña no encriptada, es decir, la contraseña que el usuario que está intentando ingresar escribió. Esta será la secuencia a verificar.

3. Pasar la secuencia a verificar por la misma función de hash que se utilizó al momento de registrar al usuario. La secuencia de bytes que retorna de la función de hash corresponderá a la secuencia encriptada a verificar.

4. Hacer la comparación entre la secuencia encriptada a verificar y la contraseña encriptada. Si son iguales, entonces la contraseña que ingresó el usuario es correcta. De lo contrario, la contraseña es incorrecta.

Para que este protocolo funcione, es necesario guardar la 'sal' aleatoria de cada usuario (en caso contrario, es imposible hacer la comparación).

5. El Juego

El juego consiste en múltiples jugadores intentando adivinar una palabra mientras otro jugador hace un dibujo para que logren adivinar la palabra[1]. Para hacer mas entretenido el juego, cada ronda consta de un tiempo en el que los jugadores deben adivinar, en caso de que se acabe el tiempo se pasa a la siguiente ronda. La duración de las rondas queda a su criterio. Una vez acabado este tiempo, o si la palabra o frase es adivinada por todos los participantes del juego, ya no se puede seguir dibujando y el puntaje es repartido. Luego de repartir el puntaje, cambia el jugador que dibuja. Basado en lo anterior, es necesario implementar un sistema de comunicación capaz de transmitir los dibujos del jugador que dibuja al resto de los jugadores[2].

5.1. Turnos

Para el correcto funcionamiento del juego, se efectuaran diversas rondas. En cada una, se seleccionará una persona al azar y le tocará dibujar la palabra que se le entregue, mientras que el resto de los participantes intenta adivinarla. Una vez que todos los participantes hayan dibujado 1 vez, se da por finalizada la partida y se determina el ganador (mayor puntaje) y el dibujo ganador (el/los que fueron adivinados por todos en la sala).

5.2. Dibujo

Se sugiere fuertemente el uso del módulo `QGraphics` de `PyQt4` para el renderizado de los dibujos en los clientes. Se deben implementar las siguientes herramientas y opciones para los dibujos:

- **Colores:** El programa debe contar con set de colores distintos para utilizar. Además, se debe dar la opción de elegir entre los 4 colores más usados por el usuario conectado. También la opción de ingresar cualquier color ingresando los 3 números RGB[3]
- **Grosor:** Se debe dar la opción de al menos 3 tipos de grosor para el dibujo.
- **Lineas:** Se debe entregar la opción de generar lineas rectas o lineas curvas[4] para facilitar la creación de un dibujo y no tener que recurrir a mano alzada.

[1] Pueden ver una vil copia de su juego en http://www.pinturillo2.com/

[2] Hint: `mouseMoveEvent` y `mouseReleaseEvent` son eventos que pueden ser útiles para determinar cuándo y qué información enviar a los jugadores.

[3] Composición del color en términos de la intensidad de los colores primarios de la luz.

[4] De la misma forma como se usa en PAINT, se selecciona el punto inicial y final... en el caso de la curva, la curvatura siempre es constante y la definen ustedes.

- **Templates:** Tantos años de usar paint sirvieron de algo. Para facilitar la creación del dibujo, se deben implementar templates de figuras geométricas pre-hechas con un tamaño standard (definido por ustedes), para usarlos en el dibujo. Para su uso, al hacer click en la opción de alguna figura, esta se debe crear en el dibujo y a continuación dar la opción de moverla a la posición deseada. Dentro de las figuras que se deben ofrecer en el programa están:
  - Circulo
  - Cuadrado
  - Rectángulo
  - Triangulo

Nota: El programa debe ser capaz de implementar combinaciones de opciones, por ejemplo, se espera que se puedan crear lineas rectas de un color y un grosor seleccionados. Queda a su criterio como implementar la selección de combinaciones en la interfaz, siempre y cuando sea de fácil uso.

5.3. Chat de Juego

Mientras un jugador esta haciendo el dibujo, el resto de los jugadores estará intentando adivinar a que palabra se refiere el dibujo, escribiendo sus respuestas en el chat. El jugador que dibuja debe tener deshabilitada la opción de hablar por chat.

El chat del juego debe tener ciertas características:

1. Deben implementar un algoritmo, para que cuando una persona este cerca de la palabra a adivinar, el programa le indique (solo a la persona) que esta cerca. Por ejemplo, si la palabra a adivinar es CASA y la persona escribe CAÑA, entonces el programa debe escribir en el chat un mensaje de que estaba cerca de la palabra, pero solo debe ser capaz de leerlo la persona que escribió.
2. Cuando a una persona le toca dibujar, escribir en el chat debe estar deshabilitado, es decir, esta persona puede leer lo que escriben los otros, pero no puede escribir nada en el chat.
3. Cuando una persona escriba la palabra correcta, no se debe mostrar en el chat de la partida, esto con el fin de que los demás participantes no copien y tengan que adivinarla por su cuenta. Ademas, el sistema debe informarle de alguna forma que adivino la palabra. Una vez adivinada la palabra, los mensajes que el jugador ganador escriba solo apareceran para otros jugadores que hayan adivinado la palabra correcta.

5.4. Puntaje

Deberán definir un sistema de puntajes, dependiendo del orden en que las personas adivinan las palabras (EJ: El primero gana 10, segundo 5, etc) y un bono para el que dibuja dependiendo de la cantidad de personas que adivinen su palabra (Premio por sus dotes artísticos). Ademas, deberán tener un dashboard que muestre los puntajes de los jugadores de la sala, para ver quien lidera, y actualizarlo al terminar cada ronda. Todo el sistema de puntajes debe quedar especificado en el README.

6. Guardar Imagen

Entre las distintas rondas del juego, cuando la imagen ya esta finalizada, se debe poder la opción de guardar la imagen en formato PNG.

6.1. Manejo de bytes formato PNG

Este tipo de imágenes, en su estructura de bytes, comienza siempre con los bytes `137 80 78 71 13 10 26 10`. Luego de esto, los bytes se ordenan en bloques o *chunks*.

Los bloques tienen una estructura establecida, que es la siguiente:

Estructura de un chunk o bloque			
Largo de información del bloque	**Tipo de bloque**	**Información**	**CRC**
4 bytes	4 bytes	Largo	4 bytes

Los primeros 4 bytes indican el largo que posee la sección de información del chunk. Los siguientes 4 bytes pueden ser decodificados para obtener un string de 4 caracteres que indican qué tipo de información posee el bloque. Luego, viene la sección de la información, que posee la cantidad de bytes indicadas en un principio. Esta información varía dependiendo del tipo de bloque en el que se esté trabajando. Por último, los siguientes 4 bytes corresponden a un código CRC (*Cyclic redundancy check*) que asegura la integridad de la información. El CRC se calcula con todos los bytes del bloque anterior, a excepción de los bytes correspondientes al largo. Para calcularlo se recomienda usar el método `crc32` de la librería `zlib`. El CRC debe ir siempre presente, incluso si el bloque no tiene información.

Tipos de bloque

- `IHDR`
  En este bloque se tiene la metadata de la imagen.

IHDR	
Cantidad de bytes	Descripción
4	Ancho en pixeles
4	Alto en pixeles
1	Profundidad de bits
1	Tipo de colores
1	Tipo de compresión
1	Tipo de filtro
1	Tipo de entrelazado

- `IDAT`
  Este bloque posee los bytes de la imagen comprimidos. En una imagen pueden existir varios bloques `IDAT`, lo que puede ser útil para hacer streaming de imágenes. En este caso particular, sólo interesa guardar localmente una imagen del dibujo, por lo que se sugiere generar un sólo bloque `IDAT`.

- `IEND`
  Este bloque indica el término del archivo, y en la sección información no posee datos (el largo de la sección es 0).

6.2. Pasar del dibujo a PNG

A continuación se explicará brevemente cómo pasar una matriz de pixeles RGB a una imagen PNG. Ante cualquier duda respecto a este proceso, y en particular, cómo construir los bloques, se recomienda revisar la especificación de PNG en el siguiente enlace: `https://www.w3.org/TR/PNG/#11Chunks`.

Creando el IHDR

En general para todos los chunks, primero se deben formar los datos. En el caso del bloque `IHDR`, a continuación se explica el significado de cada elemento que lo conforma:

- **Ancho**: El ancho de la imagen. Debe ser encodeado en 4 bytes.
- **Alto**: Igual que el punto anterior, aplicado al alto de la imagen.
- **Profundidad en bits**: Corresponde a la cantidad de bits por canal. Se recomienda usar el valor `16`. Debe ser encodeado en 1 byte.
- **Tipo de color**: Como cada pixel va a ser representado por un triple RGB, se debe usar el valor `2`. Debe ser encodeado en 1 byte.
- **Tipo de compresión**: Indica el método usado para comprimir los datos de la imagen. Actualmente sólo se encuentra definido el método con valor `0`, por lo tanto ese es el que se debe usar. Debe ser encodeado en 1 byte.

- **Tipo de filtro**: Indica el método de preprocesamiento que se aplicó a la imagen antes de comprimir los datos. Esto se usa usualmente para mejorar la performance de la compresión. Por simplicidad, se recomienda omitir el preprocesamiento. En ese caso se debe usar el valor `0`. Debe ser encodeado en 1 byte.
- **Tipo de entrelazado**: Indica el orden de transmisión de los datos de la imagen. Por simplicidad, se recomienda no usar entrelazado. En ese caso se debe usar el valor 0. Debe ser encodeado en 1 byte.

Sólo falta insertar al principio el largo de la información y tipo de bloque. Recuerda calcular el CRC y agregarlo al final del bloque. Tenemos nuestro bloque IHDR listo.

Creando el IDAT

La información descomprimida del bloque IDAT corresponde a una representación de la matriz de pixeles de la imagen. Esencialmente, cada byte corresponde a un canal del pixel que se está representando. Para ejemplificar: el primer byte representa el valor del canal rojo del primer pixel, el segundo byte el canal verde del primer pixel, el tercer byte el canal azul del primer pixel, el cuarto byte el canal rojo del segundo pixel, y así sucesivamente.

Un detalle importante es que al principio de cada fila de pixeles es necesario agregar un byte adicional que corresponde al filtro aplicado a dicha fila. Como no se aplicó ningún filtro, basta con asignarle el valor `0`. Entonces, si cada pixel estará representado por un triple RGB, y además se debe agregar un byte al principio de cada fila para el filtro, el largo de este bloque sin comprimir será $alto \times (ancho \times 3 + 1)$.

A continuación se debe comprimir el bloque de información. Para hacerlo, se recomienda usar el método `compress` de la librería `zlib`. Sólo falta insertar al principio el largo de la información y tipo de bloque. Recuerda calcular el CRC y agregarlo al final del bloque. Tenemos nuestro bloque IDAT listo.

7. Salas

La interfaz permitirá crear o unirse a Salas, donde varios amigos se juntan para disfrutar este espectacular juego. (Se necesitan mínimo 2 personas para jugar, el que dibuja y el que adivina).

7.1. Historial

Cada sala podrá contar con un registro de los dibujos que se han llevado a cabo dentro de la misma. Es decir, debe existir un menú que muestre información relevante sobre las partidas (participantes y ganadores, por ejemplo) y además, el dibujo ganador.

7.2. Invitar

La sala debe contar con la opción de invitar amigos a la sala (cualquier miembro puede invitar), si el amigo acepta ingresar a la sala, deberá esperar hasta el final de la actual ronda para incorporarse al juego

7.3. Dibujos Ganadores

La sala debe tener un registro de cuales han sido los dibujos ganadores. Entiéndase por dibujos ganadores, aquellos que fueron adivinados por todos los integrantes en el momento.

8. Amigos

Como toda plataforma social, en programillo se pueden buscar personas por el nombre de usuario. Para este sistema, al agregar a una persona a amigos, se asume que la otra persona lo acepta automáticamente, es decir, no es necesario una confirmación de ambas partes. La idea de los amigos, es que se puedan invitar a una sala (explicado mas adelante) y jugar con ellos nuevamente. Con los amigos se pueden generar chats individuales (1-1) o grupos de chat entre varios amigos, lo cual se detallará en el siguiente punto.

Ademas, se pide que se entregue un historial, en la pestaña de inicio, de aquellas personas con las que se jugo últimamente y dar la opción de agregarla a amigos (solo se deben mostrar las personas con las que jugó y que no tiene en amigos).

9. Especificaciones del Chat

Se podrán crear chats de dos o mas participantes dentro del sistema (OJO no es el mismo chat que el de la sala de juego). Para esto, la interfaz deberá permitir al usuario buscar alguna otra persona del sistema, agregarla a amigos y abrir una ventana de conversación. Este chat debe permitir las siguientes funcionalidades

9.1. Mensajería instantánea

Los usuarios podrán escribir mensajes en texto plano que se enviarán instantáneamente al otro participante.

9.2. Emojis

El chat debe dar soporte de emoticones en la conversación. Al escribir ciertas palabras clave en la conversación y enviarlas el programa deberá reemplazarlos por emoticones. Deben soportar como mínimo las siguientes palabras: `:)`, `:(`, `:o`. Quedan a su criterio los emoticones a utilizar.

9.3. Historial

Todo mensaje enviado por el chat entre dos usuarios será guardado en el servidor. Esto es para que la siguiente vez que uno de estos usuarios abra el chat, sea capaz de leer los mensajes enviados previamente.

9.4. Chat grupal

Debe existir la opción de crear un chat grupal entre múltiples usuarios (previamente seleccionados de su lista de amigos). Este chat debe contar con las mismas características que el normal, pero aplicado para múltiples participantes.

- Comenzar partida: Esta opción, permite crear una sala con los integrantes del grupo para comenzar a jugar. De haber realizado esta acción previamente, se reabre la sala donde habían jugado. De haber existido una sala, en la que ingreso una persona que no esta en el grupo de chat actual, se abre una nueva sala.
- Historial de partidas: Además del historial de texto propio de los chats, este tipo de grupos debe llevar el registro de las partidas que ha jugado dicho grupo, mostrando la información pertinente (sala, ganador), además de mostrar el dibujo ganador de cada una si se requiere.

10. Funcionamiento

Su sistema deberá ser capaz de funcionar entre computadores que estén conectados a una misma red local. Para esto es necesario que maneje de forma correcta los puertos y el host asignado a sus sockets. Para que esto funcione correctamente, y para que los ayudantes después sean capaces de corregir la tarea, **deberá seguir los siguientes pasos**.

10.1. Estructura de Archivos

El cliente y el servidor deben ser dos programas de Python que funcionen de forma totalmente independiente. Esto no implica que no pueda tener módulos en común que importe desde ambos programas. Lo importante es que el Cliente esté construido de tal forma que se pueda ejecutar en otro computador sin la necesidad de tener además el programa del servidor. Esto permitirá que se ejecute su programa desde varios computadores a la vez. Es de suma importancia que especifíque en el `README` y deje bien claro qué archivos son los que se deben tener en un computador para que el Cliente corra bien.

10.2. Servidor

Este programa no debiese necesitar una interfaz gráfica, pues este solo manejará la lógica del sistema y se correrá en un único computador siempre. En este computador es donde usted guardará los archivos "subidos" al sistema, la base de datos de los usuarios y todo lo que usted encuentre pertinente. Tambien es de suma importancia que en este archivo usted escriba las siguientes variables en las primeras líneas: `HOST` y `PORT` de tal forma que cuando el ayudante corrija su tarea, pueda escribir su propia IP y el puerto que desee utilizar como servidor. Estas son las variables que usted debiese usar en el método `bind` del socket del servidor.
Para poder realizar una conexión por red local usted solo debe hacer dos cosas

1. Estar conectado a una red
2. Setear la variable `HOST` de su servidor como la IP local de su computador. Distintos sistemas operativos tienen distintas formas de obetenerla, pero en todos es un proceso muy simple.

10.3. Cliente

Este programa debiese incluir la interfaz gráfica de `Programillo`, pues este será el programa que ejecutarán los usuarios para poder utilizar su sistema. Es importante notar que este programa no servirá de nada si es que su servidor no se está ejecutando paralelamente. Este programa debiese ser autosuficiente, de tal forma que si se lo envía a un amigo que está conectado a la misma red que su computador, y él lo ejecuta, podrá probar su sistema.

Este archivo también debe tener las variables `HOST` y `PORT` en un lugar de fácil acceso. Estas debiesen tomar el mismo valor que las del servidor nombrado en el punto anterior. Estas debiesen ser usadas en el método `connect` del socket del cliente. La idea es que el ayudante pueda modificar estas de forma expedita al momento de corregir su tarea.

11. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.5
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos `PEP8`.

- Si no se encuentra especificado en el enunciado, asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje[5] de su tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debe adjuntar un archivo `README.md` donde comente sus alcances y el funcionamiento de su sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*.
- Cree un módulo para cada conjunto de clases. Divídalas por las relaciones y los tipos que poseen en común. **Tareas que implementen el servidor o cliente en un sólo archivo serán castigadas fuertemente**.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por encima de ningún otro.

12. Entrega

- **Fecha/hora:** 20 de Noviembre - 23:59 hrs
- **Lugar:** GIT - Carpeta: Tareas/T06

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).

[5] Hasta −5 décimas.