Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2025-2)

Tarea 4

Entrega

- **Tarea y** `README.md`
  - **Fecha y hora oficial (sin atraso): lunes 24 de noviembre 20:00 horas**
  - **Fecha y hora máxima (2 días de atraso): miércoles 26 de noviembre 20:00 horas**
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T4/`.
    El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
  - **Pauta de corrección**: **en este enlace**.
  - **Bases generales de tareas (descuentos)**: **en este enlace**.
  - **Formulario entrega atrasada:** **en este enlace**. Se cerrará el **miércoles 26 de noviembre 23:59 horas**.
- **Ejecución de tarea**: La tarea será ejecutada **únicamente** desde la terminal del computador. Además, durante el proceso de corrección, se cambiará el nombre de la carpeta "`T4/`" por otro nombre y se ubicará la terminal dentro de dicha carpeta antes de ejecutar la tarea. **Los *paths* relativos utilizados en la tarea deben ser coherentes con esta instrucción.**

Objetivos

- Utilizar conceptos de interfaces y `PyQt5` para implementar una aplicación gráfica e interactiva.
- Traspasar decisiones de diseño y modelación en base a un documento de requisitos.
- Diseñar e implementar una arquitectura cliente-servidor. Además, en el cliente se debe entender y aplicar los conceptos de *back-end* y *front-end*.
- Aplicar conocimientos de *threading* en interfaces (`Thread` y/o `QThread`).
- Aplicar conocimientos de señales.
- Aplicar conocimientos de creación de redes para comunicación efectiva entre computadores (*networking*).
- Aplicar conocimientos de manejo de *bytes* para seguir un protocolo de comunicación preestablecido.

Índice

1. *DCCasino*

Han pasado años de la guerra entre el IMC y DCC, que resultó en una adquisición forzosa por parte FMAT, luego del descubrimiento de Garrakis, un planeta desértico y rico en GatoChicos de platino en polvo. Fue durante dicha adquisición que quedaste varado en el planeta en el cual el agua llega a ser lo más preciado y la arena se extiende por todo el horizonte.

En esta situación fuiste acogido por Giet, un líder nativo, de ojos azules, como todos quienes consumen en grandes cantidades GatoChico de planino en polvo. Él cree que tú eres Gisan al Gaig, es decir, "El apostador de agua", la persona capaz de cumplir el sueño de Garrakis; el agua caerá del cielo, la tierra será verde, ya nadie morirá de sed. Pero esto es un sueño que requiere de demasiada agua. Resulta que en realidad sí hay agua y está escondida bajo tierra y lo único que se necesita para cumplir el sueño es dinero.

La única forma de cumplirlo es levantando el estandarte de tu viejo empleador y construir un *DCCasino*.

1.1. Introducción

DCCasino es una experiencia en la que se podrá participar en entre 2 y 3 juegos de azar, donde lo apostado será dinero. Toda tus riquezas serán almacenadas en la base de datos, que será llevada con sumo cuidado para ir viendo las ganancias y perdidas de cada jugador a lo largo del tiempo.

Como el pueblo de Garrakis, los Gremen, están esparcidos por todo el planeta es imposible que exista un lugar físico llamado *DCCasino*, es por esto que habrá un servidor y todo gremen podrá conectarse y multiplicar su dinero para comprar agua.

Estos juegos son BlackJack, Aviator y Ruleta, estando los primeros dos detallados en Mecánicas de la partida y la Ruleta está en Bonus: Ruleta (+6 décimas). La forma de verlas está explicitado en Interfaz Gráfica e interacción. Si bien no hay un objetivo final, lo importante es divertirse y ganar dinero para recuperar el agua de los Gremen.

2. Flujo del programa

El programa comienza con la apertura de la Ventana de Inicio; en esta etapa se deberá establecer una conexión con el Servidor. Se le solicita al usuario ingresar su nombre de jugador, el cual es verificado por el servidor. En caso de que este coincida con el de otro jugador **en línea**, se debe negar el ingreso y notificar al usuario de la situación, y solicitar que intente con otro nombre. Una vez que no hay problemas con el nombre ingresado, se da acceso a la siguiente ventana.

La Ventana Principal es la interfaz que presenta un menú a través del cual los jugadores pueden realizar la mayoría de las acciones relacionadas con el programa: seleccionar un juego, unirse a una partida, cargar/retirar dinero, entre otras. El detalle de estas acciones y cómo deben manejarse se encuentra descrito en la sección Interfaz Gráfica e interacción.

Una vez que el jugador ha especificado el juego que desea jugar, deberá unirse a una partida. Si la partida puede ya estar empezada, no podrás ingresar y deberás esperar en el menú hasta que esta termine para poder ingresar. El jugador tendrá inicialmente dos opciones para jugar: Blackjack o Aviator, con posibilidad extra de incorporar una ruleta. Cada uno de estos juegos tendrán una ventana y mecánicas distintas, cada cual será explicada en las secciones de Mecánicas de Juego e Interfaz Gráfica e interacción. Es importante considerar que cada juego va a tener una sola sala, con límites de jugadores específicos para cada juego.

La Ventana Principal corresponde al espacio principal donde se desarrolla la experiencia del casino en línea, presentando el tipo de juego seleccionado correctamente: *Blackjack* o *Aviator* (o Ruleta). En esta

ventana se despliegan los elementos visuales y de interacción propios de cada modalidad: la mesa y las cartas en el blackjack, o el avión y la trayectoria de vuelo en Aviator, etc.

Sin importar el juego que se elija, se debe seguir los siguientes flujos:

Ingreso al juego
Cuando un jugador intenta ingresar a un juego, se debe revisar si la sala no está llena y un juego no está en progreso. Si no lo está, entonces el jugador puede ingresar en la sala.

- Si la ronda aún no ha comenzado y la **mesa está abierta**, puede **apostar libremente**.
- Si la ronda ya se encuentra en curso, es decir ya sucedió el cierre de mesa, entonces el jugador no puede entrar y debe esperar a que termine la ronda para poder entrar a la mesa.

Ronda de juego.
Cada ronda pasa por tres etapas:

1. **Apertura de apuestas:** los jugadores pueden realizar, modificar o cancelar sus apuestas hasta que se llegue al límite de apuestas en la mesa.
2. **Cierre de mesa:** se bloquean las apuestas y se ejecuta la lógica del juego correspondiente (reparto de cartas o vuelo del avión).
3. **Balance de resultados:** el servidor calcula las ganancias o pérdidas según las reglas del juego y actualiza los saldos de cada jugador.

Fin de ronda.
Una vez concluidos los cálculos y mostrados los resultados, el juego vuelve automáticamente al estado de **apuestas abiertas**, permitiendo iniciar una nueva ronda. Los jugadores pueden permanecer en el mismo juego, o volver al menú principal.

Durante la partida, el jugador puede observar en tiempo real las apuestas realizadas por los demás participantes, el dinero total apostado en la ronda, las ganancias obtenidas y su propio saldo disponible. Asimismo, se muestran indicadores adicionales como las otras jugadas de otros jugadores.

El jugador permanece en el juego hasta que desee salirse u ocurra una desconexión repentina del servidor. En el caso de que pierda todo su dinero en las apuestas, el jugador no podrá apostar y debe volver automáticamente al menú principal para cargar más dinero. En el caso de que ocurra una desconexión repentina, se le debe notificar debidamente al usuario y volver a la Ventana de Inicio.

3. Mecánicas de Juego

Todos los juegos comparten las siguientes mecánicas, presentes a lo largo de toda la partida.

3.1. Dinero y Apuestas

Cada jugador posee un **saldo personal** administrado por el servidor, el cual representa el monto total de dinero disponible para apostar dentro del casino, al registrarse un nuevo jugador este deberá comenzar con su saldo personal igual a `SALDO_INICIAL`[1].

El saldo puede incrementarse mediante la opción de **carga de dinero** disponible en la Ventana Principal. Asimismo, el saldo del jugador puede aumentar al ganar apuestas o reducirse al perderlas.

[1] A lo largo del enunciado encontrarás palabras escritas en `ESTE_FORMATO`. Estas corresponden a parámetros del programa. Puedes encontrar los detalles en la sección `parametros.py`.

El jugador puede apostar libremente cualquier monto que no exceda su saldo actual. Todas las transacciones que modifiquen este saldo deben ser gestionadas y validadas por el servidor.

Cada vez que el jugador confirma una apuesta, el valor correspondiente se descuenta temporalmente de su saldo y queda registrado en el servidor hasta que la ronda finalice y se muestren los resultados. Una vez declarada la **mesa cerrada** y posteriormente iniciada la ronda, no se aceptan nuevas apuestas ni modificaciones sobre las ya registradas.

En caso de que un jugador no realice una apuesta válida en un juego será enviado a la Ventana Principal para efectuar una nueva carga de dinero o seleccionar otro juego.

El sistema debe mostrar un mensaje de error para informar la razón de porque fue movido a la ventana principal.

Por otro lado, los resultados de las partidas deben ser registradas en una base de datos.

3.2. Aviator

3.2.1. Explicación del juego

Aviator es un juego donde un avion vuela y tras cierto tiempo al azar se estrella. En **Aviator**, cada jugador realiza una apuesta inicial antes del comienzo de la ronda, que debe ser como mínimo cierto valor. El avión despegará una vez se llega al máximo de apuestas permitidas. Una vez que el avión despega, comienza a aumentar un **multiplicador de ganancia** que depende directamente del tiempo transcurrido desde el inicio. Este valor, que parte en 1,00×, indica cuántas veces se multiplicará la cantidad apostada si el jugador decide retirarse antes de que ocurra el **crash**.

El **crash** corresponde al momento en que el avión se estrella, finalizando inmediatamente la ronda. Si un jugador no se ha retirado antes de ese instante, pierde por completo su apuesta. En cambio, si se retira a tiempo, su ganancia se calcula multiplicando el monto apostado por el valor actual del multiplicador.

El multiplicador crece de manera **acelerada**, aumentando con mayor rapidez a medida que el avión permanece en vuelo. Esto genera un equilibrio entre riesgo y recompensa: mientras más se espere, mayor será el posible retorno, pero también mayor la probabilidad de que el avión se estrelle y la apuesta se pierda. El objetivo es decidir con precisión el momento de retiro para maximizar la ganancia sin quedar fuera del juego.

3.2.2. Tiempo de Crash

El **tiempo de crash** corresponde al instante en que el avión se estrella y la ronda termina. Este valor debe determinarse aleatoriamente al inicio de cada partida, garantizando que el resultado sea impredecible.

Para calcularlo, se utiliza la función `betavariate(1.4, 4.0)` del módulo **random**, la cual genera un número aleatorio entre 0 y 1 siguiendo una distribución que privilegia valores bajos. Los parámetros utilizados, 1.4 y 4.0, determinan la forma de la distribución. De este modo, la mayoría de las rondas finalizarán temprano, aunque se mantiene la posibilidad de vuelos más largos de forma poco frecuente.

El valor obtenido debe multiplicarse por la `DURACION_RONDA_AVIATOR`. Así, el tiempo de crash se calcula de manera simple como:

$$t_{\text{crash}} = \text{random.betavariate}(1.4,\ 4.0) \cdot \texttt{DURACION\_RONDA\_AVIATOR}$$

El parámetro `DURACION_RONDA_AVIATOR` debe estar en segundos y por consecuencia t_{crash} también estará en esta unidad de medida.

3.2.3. Multiplicador

El **multiplicador** representa el valor que se aplica sobre la apuesta inicial del jugador y determina la ganancia potencial en cada instante de la ronda. Su valor inicial es `1.00x` y aumenta a medida que transcurre el tiempo, de modo que, si el jugador decide retirarse antes del **crash**, su ganancia será igual al monto apostado multiplicado por el valor actual de este factor.

El multiplicador de **Aviator** se incrementa de manera **acelerada**, lo que significa que su ritmo de aumento se vuelve progresivamente mayor con el paso del tiempo. Para este efecto el multiplicador se debe calcular en función del tiempo como[2]:

$$M(t) = 1 + \left(e^{0.55t} - 1\right)$$

El valor del multiplicador en cada segundo del juego debe ser calculado por el servidor y enviado a los clientes conectados en todo momento para que estos lo puedan mostrar dinámicamente en pantalla. El valor final del multiplicador alcanzado en la ronda dependerá del **tiempo de crash** determinado aleatoriamente.

3.2.4. Periodo de apuestas

Antes de comenzar cada ronda de Aviator existirá un periodo de apuestas que terminará cuando 3 jugadores hayan apostado. Cada apuesta debe ser de mínimo un monto de `APUESTA_MINIMA_AVIATOR`. Una vez suceda esto, el dinero apostado se pondrá en juego. Fuera de este periodo el jugador no podrá modificar o cancelar su apuesta y deberá esperar al siguiente **periodo de apuestas** para volver a apostar.

3.3. Blackjack

El **Blackjack** es un juego de cartas, en el cual el objetivo de cada jugador es que la suma de los valores de sus cartas se acerque lo más posible a 21, pero sin pasarse de este valor. En el juego, todos los jugadores compiten contra el *dealer* (también se le puede decir *crupier*), el cual es un representante del casino. Para ganar, cada jugador debe procurar tener un valor más alto que el *dealer*. En cada mesa, habrá 4 jugadores apostando, luego de que 4 personas apuesten comienza la ronda de juego con ellos como participantes.

3.3.1. Valores de las cartas

En Blackjack, cada carta tiene un valor específico, los cuales se presentan a continuación:

- El As puede tener un valor de 1 u 11, según le convenga al jugador en ese momento.
- Las cartas numéricas (del 2 al 10) van a tener siempre el mismo valor, correspondiente a su número.
- Las figuras (J, Q o K) tendrán siempre un valor de 10.

3.3.2. Juego

Para participar en un juego, se debe entrar con una apuesta inicial de `APUESTA_MINIMA_BLACKJACK`, la cual se realiza antes de la repartición de cartas. Luego, se realiza la repartición de las primeras 2 cartas, donde la primera será pública para todos los jugadores, y la segunda será privada, donde sólo quién recibió la carta puede verla.

Después de esto, comenzará inmediatamente la tercera repartición de cartas, la cual irá desde el jugador de más a la derecha en la mesa hasta el jugador de más a la izquierda (o viceversa), es decir, será uno por

[2]Considera el modulo math contiene constantes como euler

uno. Cada jugador tendrá la posibilidad de pedirle más cartas al *dealer* siempre que la suma de estas sea menor que 21. En el caso de serlo, perderán inmediatamente su apuesta y no podrán pedir más cartas.

Luego de haberle repartido la tercera ronda de cartas a los distintos jugadores, solo el *dealer* deberá revelar su carta oculta y entonces repartirse cartas a sí mismo siguiendo las siguientes reglas:

- En el caso de que sus cartas sumen 16 o menos, este deberá repartirse una nueva carta, y repetirlo hasta que sus cartas sumen más de 16.
- Si sus cartas suman 17 o más cartas, deberá quedarse como está, terminando así la tercera ronda de repartición.

Luego de terminada la ronda, se hacen los pagos correspondientes y se da un tiempo en donde se muestra el resultado de la partida, antes de abrir nuevamente un periodo de apuestas para participar en el próximo juego o volver a la ventana principal.

3.3.3. Repartición de cartas

Por motivos de simpleza, las cartas deberán ser aleatorias, es decir, no deben comportarse como un mazo real, sino que puede haber repetición.

3.3.4. Recompensas y pagos

Habiendo finalizado toda la repartición de cartas, el *dealer* deberá pagarle o quitarle las apuestas a los respectivos jugadores, según cómo les fue.

- Si el *dealer* se pasó de 21, entonces le pagará $\times 2$ a todos los jugadores que hayan logrado mantenerse con una suma menor o igual a 21, y le devolverá la apuesta a quienes se hayan pasado. Por ejemplo, si la *crupier* sumó 23 en total, Juan obtuvo 17 en cartas con una apuesta de 10, y Daniela obtuvo 22 en cartas con una apuesta de 10, enconces le paga 20 a Juan y le devuelve los 10 a Daniela.
- Si el *dealer* obtuvo entre 17 y 20, entonces le paga $\times 2$ la apuesta a todos los jugadores cuya suma sea mayor a la obtenida por éste, mientras que les devuelve la apuesta (paga $\times 1$) a quienes hayan empatado. Por ejemplo, si el dealer logró una suma de 19, y un jugador que apostó 10 unidades de dinero obtuvo 20 en las cartas, entonces el *dealer* le paga 20, mientras que a otro jugador que apostó 10 fichas y obtuvo 19 en las cartas, el *dealer* le paga 10.
- Si el *dealer* obtuvo un 21, le devuelve la apuesta a todos los que hayan empatado con éste y todos los otros jugadores pierden su dinero.

4. Interfaz Gráfica e interacción

Se evaluarán, entre otros, los siguientes aspectos:

- Correcta **modularización** del programa, esto quiere decir que se debe respetar y seguir una adecuada estructuración entre ***front-end*** y ***back-end***, con un **diseño cohesivo** y de **bajo acoplamiento**.
- Correcto uso de **señales** entre ***back-end*** y ***front-end***, implementación de ***threading*** cuando corresponda.
- Un flujo prolijo a lo largo del programa. Esto quiere decir que el usuario puede navegar sin problemas entre las distintas partes que componen *DCCasino* solo ejecutando una vez el programa. En otras palabras, un usuario nunca debe quedarse atorado en alguna parte del programa que lo obliga a cerrar *DCCasino* y volver a ejecutarlo. A modo de ejemplo: si un usuario ingresa a una ventana y no se puede mover a otra ventana, esto es considerado como una mala implementación.

- Generación de las ventanas mediante código programado por el estudiante. Es decir, **no se permite** la creación de ventanas con el apoyo de herramientas como `QtDesigner`, `QML`, entre otros.

Aclaración importante: los esquemas y diseños que serán expuestos son únicamente referenciales. No es necesario que tu tarea sea una copia exacta de estos: puedes agregar creatividad inventando botones nuevos, interacciones nuevas, diseños y hasta animaciones. Sin embargo, será evaluado que los elementos **mínimos** estén presentes y funcionen del modo que se exige (detallado más adelante).

Estos elementos mínimos se explicitan en cada sección y hacen referencia a comportamientos esperados con la interacción, y a información que debe ser mostrada. A menos que se explicite por medio del nombre de un *widget* específico, eres libre de mostrar la información y crear la interacción por medio de los componentes que desees, ya sean inventados por tí o propios de la librería `PyQt`.

A pesar de lo anterior, en ningún caso se les exige cosas relacionadas a la "belleza" de la interfaz gráfica. Por lo tanto, una ventana que sólo tenga los elementos expuestos en los esquemas, tiene el mismo nivel de validez que otra llena de decoraciones y efectos interactivos, esto, suponiendo que ambas tengan el comportamiento deseado.

Finalmente cabe destacar que si se desea desarrollar funcionalidades extras, estas serán evaluadas bajo los mismos criterios generales mencionados anteriormente. Esto quiere decir que, si el programa falla debido a una funcionalidad extra, se aplicará el descuento en el ítem correspondiente.

4.1. Ventana de Inicio

Esta es la primera ventana que se debe mostrar al ejecutar el programa. Su función principal es solicitar al usuario que ingrese el nombre de jugador que desea utilizar. Debe contener al menos los siguientes elementos:

- **Nombre del programa**: debe presentarse el nombre del juego en la ventana de inicio.
- **Editor de línea** para que el usuario pueda ingresar el nombre de jugador.
- **Botón de ingreso** para solicitar ingresar con el nombre indicado.

En la figura 2 se muestra un ejemplo esquemático de los elementos mínimos solicitados con *(A)*: nombre del programa, *(B)*: editor de línea y *(C)*: botón de ingreso.

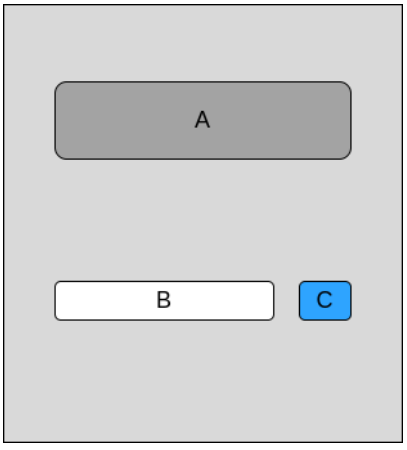


Figura 1: Esquema de ventana de inicio.

Además, en la ventana de inicio se debe mostrar la información relacionada a la conexión con el servidor y la aceptación del nombre de jugador. Esto se puede lograr agregando un elemento como un texto en

la ventana, a través de un mensaje de diálogo, etc. Lo importante es que se muestre al usuario el estado pertinente con la indicación de cómo proceder.

Específicamente, el párrafo anterior se refiere a que, si el servidor no está accesible (no se puede conectar), se debe indicar la situación y sugerir que intente conectarse nuevamente. Si el nombre de jugador no es aceptado, se debe solicitar que ingrese un nuevo nombre de jugador. Si una vez se ha abandonado la ventana de inicio y se ha entrado a la ventana principal o a la ventana de juego, y se pierde la conexión con el servidor, se debe informar en la consola y cerrar todas las ventanas del *DCCasino* automáticamente.

4.2. Ventana Principal

Esta interfaz se mostrará si la conexión al servidor fue exitosa y se ha ingresado con un nombre de jugador válido. En esta ventana se pueden realizar diferentes acciones, cada una de ellas debe estar asociada a un elemento de la interfaz que permita al usuario llevar a cabo dicha acción.

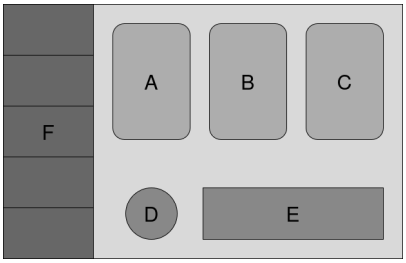


Figura 2: Esquema de ventana principal.

Antes de describir las acciones, expliquemos como se deben mostrar las informaciones en:

(E) : Deben ser visibles tanto el nombre como el saldo actual del usuario cuya sesión está iniciada.

(F) : Deben mostrarse las últimas 5 ganancias y/o pérdidas globales de todos los jugadores.

Las acciones se describen a continuación.

4.2.1. Ingresar a un juego

Se deben tener entre hasta 3 botones, *(A, B, C)*, uno por cada juego implementado, que al ser presionados abrirán una ventana con el juego elegido, siempre y cuando este se encuentre en etapa de apuestas abiertas. En otro caso, se debe impedir que el usuario entre a la sala y notificarle de la razón. En caso de que no hayan implementado un juego pueden dejar esa área vacía o pueden dejar un placeholder para ocupar el espacio.

4.2.2. Cargar dinero

Un botón en la esquina inferior izquierda, *D* que debe decir "cargar". Una vez se interactua con él, se debe abrir la ventana de recargar.

4.3. Ventana de juego BlackJack

En esta ventana podrás jugar Blackjack y ver el estado de la partida.

De forma que en la visualización, *(A)*, sea visible la cantidad que has apostado.También se debe ver la información del turno, es decir, quién está jugando, además de poder interactuar con *(E)*, *(F)*, *(G)* y *(H)*.

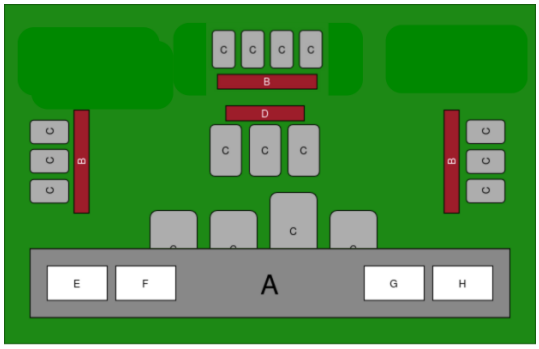


Figura 3: Esquema de Ventana de BlackJack.

Las cartas deben ser visibles, esto es, su imágen debe estar visible en el tablero. Las cartas del jugador estarán "bajo" la sección *(A)*. Las cartas privadas de los otros jugadores deben usar la imágen de una carta boca abajo. Puedes ver los *assets* (imágenes que puedes usar para las cartas) de las cartas en Cartas.

Con lo que respecta a las acciones

(E) Botón para pedir cartas cuando sea el turno del usuario, al darle *click* se debe añadir otra carta a tu mano.

(F) Botón para apostar. Al interactuar se le debe pedir al usuario el monto que desea apostar para entrar a la partida.

(G) Botón para volver a la ventana principal.

(H) Botón para salir del *DCCasino*.

Todos estos botones se deben regir bajo las reglas descritas en Blackjack.

4.4. Ventana de juego Aviator

La **ventana de juego de Aviator** corresponde a la interfaz principal donde se desarrollan las rondas del juego y se gestionan las apuestas de los jugadores en tiempo real. Su diseño se divide en tres zonas claramente diferenciadas, identificadas en las Figuras 4 y 5:

- **(A) Barra de jugadores:** ubicada al costado izquierdo, muestra a todos los jugadores conectados junto con su información de apuesta. Cada fila contiene el *nombre del jugador*, el *monto apostado*, el *multiplicador de retiro* (que permanece vacío hasta que el jugador se retire) y la *ganancia obtenida*. Esta lista se actualiza dinámicamente a medida que los jugadores apuestan o retiran su dinero. En la parte inferior de esta barra se incluye un recuadro que indica el **estado actual de la ronda** (Periodo de apuestas o Ronda en curso) y un temporizador que muestra el tiempo restante o transcurrido, según corresponda. Finalmente tiene un botón para volver al menú principal, cuando los jugadores deseen hacerlo.
- **(B) Área de juego:** ocupa la parte superior derecha de la ventana. Durante el Periodo de apuestas, se muestra un texto central indicando que las apuestas están abiertas (Figura 4). Una vez iniciada la ronda, el área despliega una animación con el **avión despegando** y una **curva creciente** que representa el aumento del multiplicador con el tiempo. En el centro de esta zona se destaca en gran tamaño el valor actual del multiplicador, que se actualiza en tiempo real hasta que ocurre el *crash*. [3]

[3]Puedes graficar un punto en PyQt ocupando la funcionalidad de drawPoint. Podrías ir colocando puntos o *labels* por donde ha pasado el avión para simular la curva del gráfico.

- **(C) Área de apuestas:** ubicada en la parte inferior derecha, permite al jugador ingresar el **monto a apostar** y realizar las acciones correspondientes mediante un único botón multifunción. Este botón cambia su texto y comportamiento según la fase del juego:
  - Durante el Periodo de apuestas: muestra **Apostar (Monto)**. Si el jugador ya apostó, se bloquea el botón y no puede cambiar el monto.
  - Al iniciar la Ronda en curso: para los jugadores que mantuvieron una apuesta activa, el botón cambia automáticamente a **Retirar (Monto)**. En este estado, el monto mostrado corresponde a la ganancia potencial, calculada como *Monto apostado* × *Multiplicador actual*, y se actualiza en tiempo real.
  - Si el jugador no realizó una apuesta durante el periodo anterior, el jugador es enviado a la ventana principal inmediatamente.

  Además, esta sección muestra el **saldo actual del jugador**, incluye un recuadro para ingresar el dinero a apostar y, debe incluir mensajes de validación (por ejemplo, saldo insuficiente o monto inválido).

En la Figura 4 se observa la interfaz durante el **Periodo de Apuestas**, mientras que la Figura 5 muestra su estado en una **Ronda en curso**. Ambas muestran la estructura general de la ventana principal de juego; recordar además, incluir en algún lugar de esta ventana un botón para regresar al menú principal.

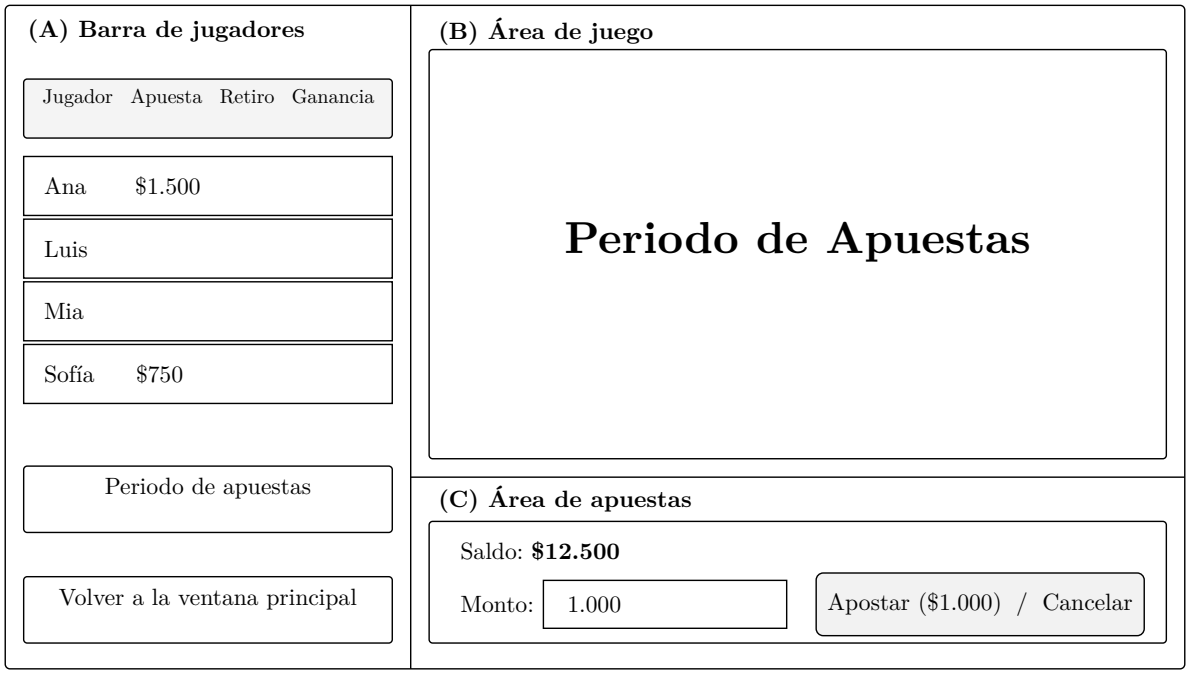


Figura 4: Esquema ventana de juego Aviator — *Periodo de Apuestas.*

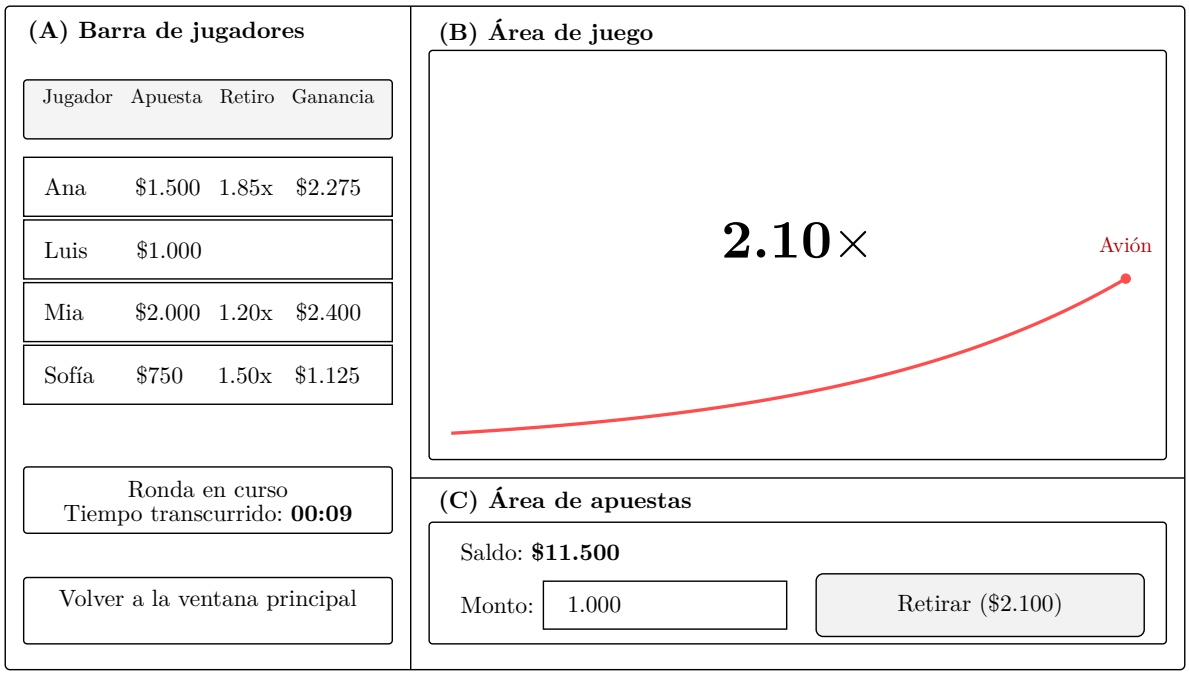


Figura 5: Esquema ventana de juego Aviator — *Ronda en curso.*

5. Bonus: Ruleta (+6 décimas)

Esta tarea posee un bonus correspondiente a implementar un tercer juego, llamado Ruleta. Para optar al bonus, deberás cumplir los siguientes requisitos:

- Haber implementado completamente los dos juegos anteriores (Blackjack y Aviator).
- Tener nota igual o mayor a 4.0 en el resto de la tarea.
- Realizar el flujo completo de este juego bonus (es decir, no solo implementar la fase de apuestas, o solo la de ejecución del juego).

5.1. Ruleta

El juego de la ruleta consiste en una rueda que contiene 37 números igualmente espaciados, los cuales están relacionados a un color en específico (rojo o negro y un 0 verde) que se encuentran intercalados. A esta combinación número color le llamaremos **casilla**. Esta ruleta dará valores totalmente aleatorios, y dependerá sólo del servidor. Para un juego de ruleta, cuando apuestan 4 jugadores comienza la ronda.

5.2. Mecánicas de la ruleta

5.2.1. Periodo de apuestas

El jugador, por medio de la interfaz, deberá introducir 2 valores: el monto de la apuesta, el cual deberá ser mayor o igual a `APUESTA_MINIMA_RULETA`, y la casilla por la cual desea apostar. El jugador puede apostar en favor de un color (negro, rojo o verde) o un número (del 0 al 36 incluyendo ambos extremos). El jugador puede realizar sólo una apuesta por ronda (ya sea número o color) y a final de la ronda se le debe recompensar a cada jugador de manera acorde.

Si el jugador no alcanza a realizar la apuesta antes de que se llegue al límite de la sala, el programa comenzará la ronda y enviará al jugador a la ventana principal.

Las recompensas por cada casilla se muestran en la tabla 1.

Casilla	Multiplicador de apuesta
Rojo o negro	2
Verde	36
Número solo	36

Cuadro 1: Recompensas de la ruleta, según la apuesta realizada

5.2.2. Giro de la ruleta

Cuando todo esto esté listo, la bolita de la ruleta comenzará a girar alrededor de esta, la cual deberá moverse según una función del ángulo, la cual está dada por:

$$\theta(t) = 360° \cdot f \cdot t$$

donde f es un parámetro que corresponde a `FRECUENCIA_GIRO_RULETA`, definido previamente, y t corresponde al tiempo en segundos que han pasado desde que empezó a girar. La bolita deberá detener su giro cuando haya pasado un tiempo aleatorio, que fluctúa entre 2 y 5 segundos

Cuando la ruleta deje de girar, se debe recompensar a los jugadores respecto al color y el número de la casilla que salió, y se debe notificar las ganancias y pérdidas, tal como se especifica en Interfaz Gráfica e interacción. El número elegido será el que se encuentre al lado de esta bolita (es decir, en el mismo ángulo). Esta imágen estará alineada perfectamente en su ángulo considerando la parte superior como el ángulo 0° (ver Interfaz Gráfica e interacción para un ejemplo)

5.3. Ventana de juego Ruleta

La ventana de juego de la ruleta deberá contener, como mínimo, los siguientes elementos (figura 6): La ruleta, un área que incluya monto y casilla, además de una opción para elegir si seguir en el juego, Barra de jugadores en donde se incluyan los nombres de los jugadores y sus apuestas (monto y casilla) y botón para volver a la página principal.

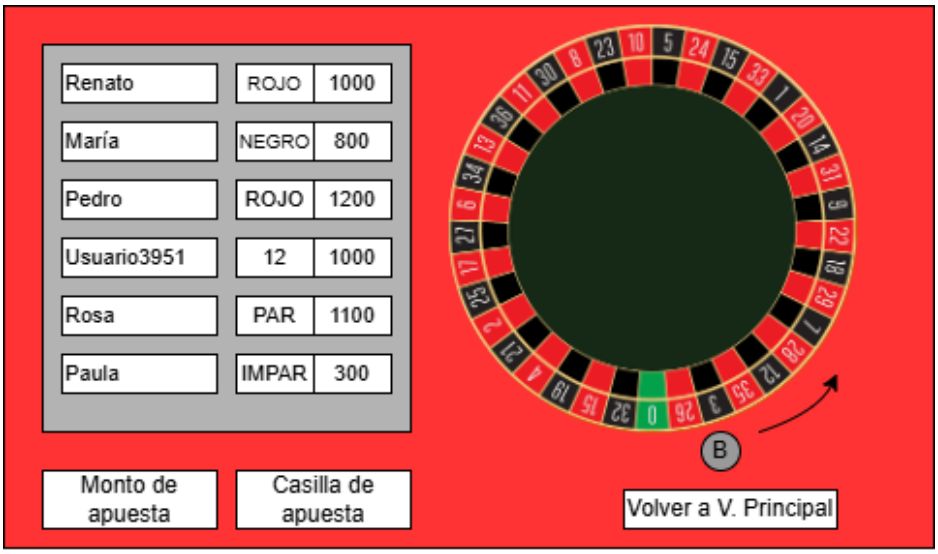


Figura 6: Ejemplo de la interfaz de la ruleta. La B corresponde a la bolita que gira alrededor de la ruleta.

5.3.1. Representación y movimiento de la ruleta

La ruleta deberá quedarse quieta, y debe ser la bolita la que gira alrededor de ella. Para esta bolita, se debe hacer un círculo, el cual se moverá alrededor de la ruleta con los números. La distancia desde el centro de la ruleta y la bolita debe mantenerse constante, y la velocidad de esta también (véase Giro de la ruleta). La posición de la pelota debe actualizarse al menos cada 0.5 segundos, pero puede ser más rápido si lo deseas.

5.3.2. Comunicación de los resultados

Al final de cada ronda, cuando haya salido una casilla y se deba recompensar a cada uno de los jugadores, la interfaz deberá incluir un mensaje con los resultados en algún lugar de la ventana, que dure unos segundos, luego de esto comienza la siguiente ronda de ruleta.

6. Arquitectura cliente-servidor

En esta sección se describe detalladamente la arquitectura que se espera utilices para desarrollar *DCCasino*. Esto hace referencia a la estructura de las entidades que interactúan, es decir, cómo cada cliente debe manejar su existencia respecto del servidor del juego, y cómo este último debería administrar los clientes y eventos relacionados al juego.

DCCasino permite múltiples jugadores por cada partida. Para lograr esto, el comportamiento del juego y la conexión de los clientes se realiza a través de un servidor que debe manejar correctamente todos los eventos de cada cliente.

El cliente y el servidor son dos entidades completamente independientes que simulan encontrarse en computadores distintos. Es por ello que el primero no puede acceder a los recursos del otro, ni viceversa. En términos prácticos, el cliente y el servidor deberán estar desarrollados en directorios distintos. Cada directorio debe contener los recursos necesarios para que la entidad presente pueda ejecutarse sin dificultades, y sin acceder al directorio de otra entidad.

Un ejemplo de cómo se realiza esta separación se muestra en la figura 7. Se puede observar que cada directorio está separado y ambos tienen su propio archivo principal `main.py` que debe ser ejecutado. Notar que estos son los únicos dos archivos que se ejecutarán.

Además el servidor debera contener la **base de datos**, especificada en *Base de datos*.

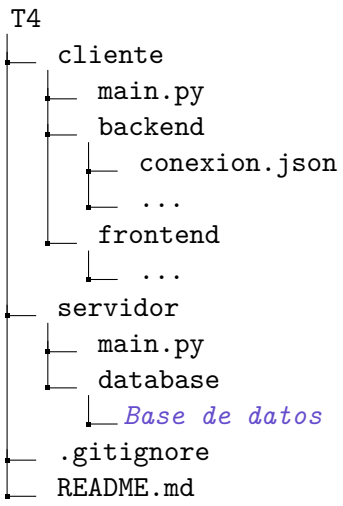


Figura 7: Organización mínima del directorio

6.1. Conexión *networking*

Cada cliente debe utilizar *sockets* para establecer una conexión al servidor siguiendo el protocolo **TCP/IP**. El servidor, por su parte, deberá levantar un puerto al cual los clientes puedan conectarse. La conexión levantada en tal puerto también debe seguir el protocolo **TCP/IP** y ser creada utilizando *sockets*.

Para lograr la conexión, es necesario que el cliente intente conectarse justamente al *socket* que el servidor ha expuesto para ese propósito. Por ello deberás crear, tanto en el **servidor** como en el **cliente**, un archivo `conexion.json` que contenga toda la información relevante para establecer la conexión de *networking* y para levantar la **API** descrita en *WebServices*. El servidor utilizará la información de este archivo para levantar un puerto, y el cliente para conectarse a dicho puerto. El archivo debe tener la siguiente estructura:

```
{
    "host": <direccion_ip>,
    "puerto": <puerto>,
    "puertoAPI": <puerto>,
}
```

6.2. Codificación y encriptación

La información que se comunica entre el cliente y servidor debe estar **codificada** y **encriptada**. Esta condición de comunicación se aplica para la información que ambas entidades envían y reciben, es decir, toda comunicación de *networking* debe seguir el protocolo de codificación y encriptación aquí descrito.

Cuando el cliente o servidor desean enviar un mensaje, el contenido de este primero debe ser serializado y transformado en *bytes*. El resultado de esta serialización es el **objeto** que se está enviando, y el total de *bytes* es su largo.

Una vez se tiene el objeto, se debe dividir en segmentos ordenados de 124 *bytes* denominados *chunks*. Si el número de *bytes* del objeto no es divisible por 124, se debe extender el final del objeto hasta que lo sea, de modo que se puedan crear sin problemas todos los *chunks*. Esta extensión debe realizarse concatenando tantos *bytes* cero (`b"\x00"`) como sean necesarios.

Luego de tener los *chunks*, se deben enumerar ordenadamente con un contador de 4 *bytes* en formato *big endian*. El número asignado a cada *chunk* debe ser concatenado al inicio del mismo. Este arreglo de *bytes* de número identificador concatenado con el contenido del *chunk* se denomina **paquete**.

En este paso se debe **encriptar** cada paquete. Para ello se deberá realizar una operación `XOR`[4] por cada *byte* del mensaje utilizando una clave de 128 *bytes* llamada `CLAVE_CASINO` [5]

La composición final del mensaje debe incluir al inicio 4 *bytes* que indiquen el largo del **objeto** que está siendo enviado. Estos *bytes* deben estar en formato *little endian*. Esto permitirá al receptor reconstruir el objeto tras ordenar los paquetes y desencriptar utilizando la `CLAVE_CASINO`. En la figura 8 se puede observar un diagrama que ilustra cómo se debe componer el mensaje enviado.

6.3. Servidor

El servidor es el controlador central de todo el juego. En este se administra la conexión con los clientes, la agrupación de jugadores y creación de partidas y toda acción relevante respecto al flujo del juego y la producción de sus mecánicas, incluyendo la interacción con la API.

[4] En Python puedes hacerlo así.
[5] Recibirás una de ejemplo en *parametros.py*

Figura 8: Ejemplo de mensaje completo

6.3.1. Mecánicas de la partida

Como se explicó en Mecánicas de Juego, es deber del servidor determinar el comportamiento de la partida. Esto quiere decir que es el servidor quien debe administrar y mantener todos los valores que definen la partida y su comportamiento. Los valores que tiene el servidor son los únicos válidos que se deben considerar para los puntajes, dinero y demases. Entre ellos se incluyen:

- La apuesta mínima para todos los juegos
- Los parámetros de cada partida dependiendo cuál es el juego
- El saldo de cada jugador.
- Cualquier elemento de un juego que deba ser sincronizado entre jugadores.

Todas estas cosas deben ser manejadas desde el servidor, y los resultados de los cálculos del servidor deben ser enviados a cada cliente para que estos puedan reflejar en su interfaz los eventos de la partida.

Cuando una partida termina, el servidor debe interactúar con la **API** descrita en *WebServices* para actualizar los datos de usuarios y en general guardar registros asociados.

6.3.2. Desconexión repentina

En el caso de que el cliente se desconecte, ya sea por error o a la fuerza, tu programa debe ser capaz de manejarlo.

Si un cliente se desconecta, deberá aparecer un mensaje en la terminal del servidor informando la situación. Este mensaje debe incluir el nombre del Cliente que se ha desconectado. A continuación, el Servidor será el responsable de eliminar su conexión.

En el caso de que la desconexión ocurra mientras el Cliente participa en una partida, será **considerado** su ganancia/perdida de dinero en todo el juego hasta que se desconectó y será eliminado de la sala. Debe además perder el dinero que tenía apostado, mientras que los demás jugadores pueden continuar con la partida.

Por otro lado, en caso de que la desconexión sea del servidor, entonces todas las apuestas son retornadas a los jugadores (no pierden ni ganan dinero), se les debe informar a los jugadores en consola y luego debe cerrarse su aplicación totalmente.

7. *WebServices*

Cuando se inicia el servidor, además de abrir los puertos necesarios para que los clientes se conecten, deberá levantar una **API** que ofrezca diferentes *endpoints* para funcionalidades distintas.

Algunos de estos *endpoints* estarán disponibles públicamente sin la necesidad de autenticación, mientras que otros requerirán el uso de un token de autenticación para validar la acción asociada. Para la auten-

tication, se debe ocupar el *header* de la *request* usando el valor de `TOKEN_AUTENTICACION` que solo el servidor debe tener.

Esta *API* debe levantarse en el *host* y *port* definidos para la API en el archivo `.json` que se describe en Conexión *networking* . Deberá estar disponible en todo momento mientras el servidor se esté ejecutando, ya que es el servidor el que interactúa con la API. **No se debe** crear un archivo de entrada (`main.py`) distinto para esta **API**, sino que el servidor debe ser capaz de ofrecer los *endpoints* ejecutando solo un `main.py`, el cual además se encarga de todo lo descrito en Servidor. Deberás usar **Threads** para manejar esto. Este *WebService* es la entidad que lee y modifica la información de los archivos en la *Base de datos*.

Los archivos con los que trabaja la **API** son exclusivos del servidor. Solo esta entidad tiene acceso a ellos tanto de lectura como de escritura.

Se espera que uses los parámetros de `json.load()` y `json.dump()`, para que así puedas serializar y modificar de una forma más cómoda tus datos. Por otro lado, el retorno de cada *endpoint* debe ser del tipo `flask.Response`.

Los *endpoints* **mínimos** que debe tener esta **API** se describen a continuación. En este listado, si un *endpoint* se indica como privado, deberás integrar la autenticación necesaria para asegurar que solo el servidor sea capaz de interactuar con este. Durante el funcionamiento correcto de la aplicación se espera que todos los endpoints de la API serán utilizados **al menos una vez**.

***(público)* GET /users/:id** Este *endpoint* es utilizado por el servidor para verificar si un usuario ya existe en la base de datos. Obtiene el **saldo actual** de este usuario si existe, necesario para el flujo del sistema. El usuario a buscar se obtiene con la ruta que se accede, siendo "id" el nombre del usuario que se busca.

Header: No debe incluir información de autenticación, puede ir vacío.

Body: No debe incluir.

$\Longrightarrow$ **Retorna:** La información del usuario, si existe, o un mensaje de error en caso contrario.

***(privado)* POST /users** Este *endpoint* es utilizado por el servidor cuando un usuario del casino entra por primera vez . Registra la información del usuario en la base de datos, incluyendo su nombre, y el **dinero disponible inicial**.

Header: Debe incluir información de autenticación, en específico, el token mencionado anteriormente.

Body: Debe contener la información inicial del usuario en formato JSON (nombre y dinero).

$\Longrightarrow$ **Retorna:** Un código de éxito o error según corresponda. En caso de éxito se retorna el código 200, en caso de error 400.

***(privado)* PATCH /users/:id** Este *endpoint* permite **actualizar la información de un usuario existente**, explícitamente su saldo. Es utilizado por el servidor durante el flujo normal de uso o al cierre de sesión, considerando la sesión en la que se encuentra además.

Header: Debe incluir el *token* de autenticación.

Body: Debe contener los campos a actualizar en formato JSON. En específico su saldo nuevo.

$\Longrightarrow$ **Retorna:** Confirmación de actualización o un mensaje de error.

***(público)* GET /games?n=N** Este *endpoint* es utilizado por el servidor para obtener las últimas `N` ganancias o pérdidas resultantes de una partida, obtenidas a partir del archivo 'ganancias.csv'. En caso de no presentar el parámetro opcional `N`, por defecto retornará las últimas 3.

Header: No debe incluir información de autenticación, puede ir vacío

Body: No debe incluir.

$\Longrightarrow$ **Retorna:** Una respuesta con la lista de los últimos `N` registros de ganancias o un código de error según corresponda.

(privado)* POST /games/*:juego_específico Este *endpoint* es utilizado por el servidor para **guardar la información de una partida** correspondiente a un juego específico. Debe actualizar el archivo ganancias, siguiendo su formato apropiado.

Header: Debe incluir el *token* de autenticación.

Body: Debe incluir los datos de la partida, como identificador del juego, jugadores, resultados, puntajes, etc.

$\Longrightarrow$ **Retorna:** Un código de éxito o error según corresponda.

8. *Base de datos*

Aquí se describen los archivos que se deben utilizar para registrar toda la información del juego, mismos que deben contener timestamps en Unix time.

Los archivos mínimos son:

1. `usuarios.csv`: en cada línea de este archivo se encuentra la información del nombre de un usuario, la fecha en que se creó la cuenta para ingresar al *DCCasino* y su saldo actual. En caso de que el archivo no exista, el programa debe poder crear el archivo desde cero y rellenarlo correctamente.

```
PedroPicapiedra,1431255927.7087214,0
Jhonny Depp,712345678.1234567,21330
Jhonny Bravo,1501239876.8192765,9993
rogueMaster,1108764312.5543981,15
```

2. `ganancias.csv` : en cada línea se debe tener, identificador de la partida, nombre de usuario, *timestamp* de la salida en formato Unix Time y el monto ganado o perdido con respecto al usuario y partida. Al igual que el archivo anterior, el programa debe poder crear este archivo desde cero en caso de que no exista.

   En el archivo de ejemplo, la línea 1 muestra el caso en que no se alcance a retirar el dinero en la séptima ronda de Aviator, pero puede existir el caso que sí, en el cual se debe indicar el momento del retiro.

   La primera columna sirve para identificar de qué ronda de qué juego está saliendo la información, es simplemente una letra que representa al juego, siendo el prefijo A para aviator, B para blackjack, R para ruleta y P para depósito de dinero. Luego está el nombre de usuario del jugador que tuvo esa ganancia o pérdida, para poder identificarlo. Luego hay un *timestamp*, que es el final de la partida (es decir, cuando se otorgan las ganancias). . Finalmente está la ganancia o pérdida que tuvo el jugador, desde el punto de vista del jugador.

   En el ejemplo de más abajo se pueden ver los resultados de varias partidas. En primer lugar una partida de aviator donde LaLuDoNoExistePatia perdió 8828 unidades de dinero. Luego hay dos partidas de ruleta distintas, donde los jugadores ganaron 2782 en dos partidas distintas. Finalmente hay un depósito de Ignacio de 3210 unidades de dinero.

```
A,LaLuDoNoExistePatia,1761256072.6645644,-8828
R,Ralsei,1759243917.132011,2782
R,Bodoque,1762248510.0003,2782
B,AgentCooper,1761259962.974682,3928
P,Ignacio,1821246933.3758185,3210
```

9. Archivos

En esta sección se describen los recursos del juego. Estos recursos son archivos que pueden pertenecer al cliente o al servidor, y son utilizados para crear una mejor experiencia de juego. Estos sprites entregados deben ser ignorados al momento de entregar tu tarea mediante el `.gitignore`. Sin embargo, si lo deseas puedes usar tus propios sprites o añadir adicionales. En dicho caso, cualquier imagen que no se haya entregado por defecto sí debe ser subida con tu entrega.

9.1. *Sprites*

Estos son archivos de imagenes que deben utilizarse para crear una mejor experiencia visual para el usuario, y para poder aplicar algunas mecánicas esenciales de *DCCasino*.

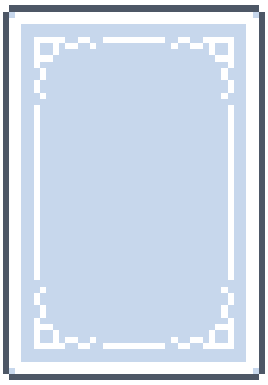
9.1.1. Aviator

Esta es la imagen del avión que deben utilizar para representar al avión en vuelo en el juego Aviator, la que van a recibir en formato `png` y `svg`, pueden utilizar cualquiera de las dos. La imagen se puede ver a continuación:

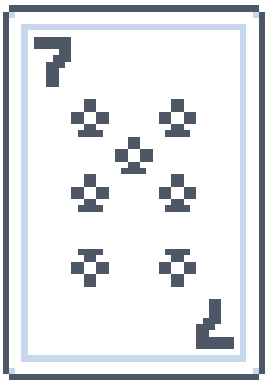
9.1.2. Cartas

Esta es la representación de las Cartas de Blackjack. Cuentas con 55 *assets* de cartas, incluyendo tréboles, picas, corazones y diamantes, además de una carta boca abajo. El nombre de los archivos sigue el siguiente formato: `cards_PINTA_VALOR.png`, donde `PINTA` puede tomar los siguientes valores: `clubs`, `diamonds`, `hearts`, `spades` y `VALOR` va de 02 a 09, y `A, J, K, Q`. Todas estas imágenes las recibirán en formato `png`.

Algunos ejemplos son:



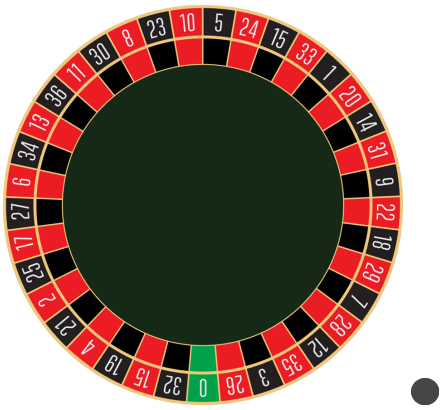
(a) Carta boca abajo



(b) 7 de tréboles

9.1.3. Ruleta

Contarán con la siguiente ruleta para poder implementar la vista del juego homónimo. La imagen la recibirán en formato `png` y también tendrán acceso a un pequeño círculo en formato `png` para representar la bolita.



9.2. `parametros.py`

Para esta tarea, se requiere la creación de un archivo `parametros.py` **tanto en la carpeta del `cliente/` como en la del `servidor/`.** En este deberás **completar todos los parámetros mencionados a lo largo del enunciado**. Dichos parámetros se presentarán en `ESTE_FORMATO` y en ese color. Además, es fundamental incluir cualquier valor constante necesario en tu tarea, así como cualquier tipo de *path* utilizado.

Un parámetro siempre tiene que estar nombrado de acuerdo a la función que cumplen, por lo tanto, asegúrate de que sean descriptivos y reconocibles. Un ejemplo de parametrizaciones es la siguiente:

```
CINCO = 5 # mal parámetro
PROBABILIDAD_CORRUPCION = 0.2 # buen parámetro
```

Si necesitas agregar algún parámetro que varíe de acuerdo a otros parámetros, una correcta parametrización sería la siguiente:

```
PI = 3.14
RADIO_CIRCUNFERENCIA = 3
```

```
3 AREA_CIRCUNFERENCIA = PI * (RADIO_CIRCUNFERENCIA ** 2)
```

Dentro del archivo `parametros.py`, es obligatorio que hagas uso de todos los parámetros almacenados y los importes correctamente. Cualquier información no relacionada con parámetros almacenada en este archivo resultará en una penalización en tu nota. Recuerda que no se permite el ***hard-coding***[6], ya que esta práctica se considera incorrecta y su uso conllevará una reducción en tu calificación.

10. `.gitignore`

Para esta tarea **deberás utilizar un** `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T4/`.

Los elementos que no debes subir y **debes ignorar mediante el archivo** `.gitignore` para esta tarea son:

- El enunciado.
- El archivo `README_inicial.md` (no confundir con el archivo **obligatorio** `README.md`)
- Cualquier archivo *bytecode*, es decir, cualquier archivo con extensión `.pyc`
- Cualquier carpeta que almacene archivos de tipo dato o imágenes que no sean los entregados junto al enunciado de la tarea (los especificados en esta parte Archivos)

Recuerda **no ignorar archivos vitales de tu tarea como los que tú creas o modificas, o tu tarea no podrá ser revisada**.

Es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos **deben** no subirse al repositorio debido al uso correcto del archivo `.gitignore` y no debido a otros medios.

[6] *Hard-coding* es la práctica de ingresar valores directamente en el código fuente del programa en lugar de parametrizar desde fuentes externas.

11. Importante: Corrección de la tarea

En el **siguiente enlace** se encuentra la distribución de puntajes. En esta señalará con color **amarillo** cada ítem que será evaluado a nivel funcional y de código, es decir, aparte de que funcione, se revisará que el código esté bien confeccionado y que la funcionalidad esté correctamente integrada en el programa. En color **azul** se señalará cada ítem a evaluar el correcto uso de señales para la comunicación *front-end*/*back-end*. Todo aquel que no esté pintado de amarillo o azul, significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea.

Importante: Todo ítem corregido por el cuerpo docente será evaluado únicamente de forma ternaria: cumple totalmente el ítem, cumple parcialmente o no cumple con lo mínimo esperado. Finalmente, todos los descuentos serán asignados manualmente por el cuerpo docente respetando lo expuesto en **el documento de bases generales**.

La corrección se realizará en función del último *commit* realizado antes de la fecha oficial de entrega (lunes 24 de noviembre a las 20:00). Si se desea continuar con la evaluación en el periodo de entrega atrasado, es decir, realizar un nuevo *commit* después de la fecha de entrega, **es imperante responder el formulario de entrega atrasada** sin importar si se utilizará o no cupones. Responder este formulario es el mecanismo que el curso dispone para identificar las entregas atrasadas. El enlace al formulario está en la primera hoja de este enunciado y estará disponible para responder hasta el miércoles 26 de noviembre a las 23:59.

Para terminar, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante de Bienestar de tu sección. El correo está en el **siguiente enlace**.

12. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.12.9.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py` que estén correctamente ordenados por carpeta. **No se revisará archivos en otra extensión como `.ipynb`**.
- Toda el código entregado debe estar contenido en la carpeta y rama (*branch*) indicadas al inicio del enunciado. Ante cualquier problema relacionado a esto, es decir, una carpte distinta a `T4` o una rama distinta a `main`, se recomienda preguntar en las ***issues* del foro**.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un **único archivo markdown**, llamado `README.md`, **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. El no incluir este archivo, incluir un readme vacío o el subir más de un archivo `.md`, conllevará un **descuento** en tu nota.
- Esta tarea se debe desarrollar **exclusivamente** con los contenidos liberados al momento de publicar el enunciado. No se permitirá utilizar contenidos que se vean posterior a la publicación de esta evaluación.
- Se encuentra estrictamente prohibido citar código que haya sido publicado **después de la liberación del enunciado**. En otras palabras, solo se permite citar contenido que ya exista previo a

la publicación del enunciado. Además, se encuentra estrictamente prohibido el uso de herramientas generadoras de código para el apoyo de la evaluación.

- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro que sí sea especificado por enunciado.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0)**.**