



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

IIC2233 Programación Avanzada (2025-1)

## Tarea 4

### Entrega

- **Tarea y README.md**
  - **Fecha y hora oficial (sin atraso):** lunes 23 de junio 20:00 horas
  - **Fecha y hora máxima (2 días de atraso):** miércoles 25 de junio 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 25 de junio 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

<b>1.</b> <i>DCCaída de palabras</i>	<b>4</b>
1.1. Introducción . . . . .	4
<b>2.</b> <i>Flujo del programa</i>	<b>4</b>
<b>3.</b> <i>Mecánicas de Juego</i>	<b>5</b>
3.1. Tiempo de juego . . . . .	6
3.2. DCConjuntos de palabras . . . . .	6
3.3. Aparición de palabras . . . . .	6
3.4. Ingreso de palabras . . . . .	7
3.5. Explosión de palabras . . . . .	7
3.5.1. Racha . . . . .	7
3.6. Supervivencia: ganar o morir . . . . .	7
3.7. Dificultad . . . . .	8
3.7.1. Intervalo de aparición . . . . .	8
3.7.2. Tiempo de caída . . . . .	8
3.8. Bloques de palabra . . . . .	9
3.8.1. Bloque de hielo . . . . .	9
3.8.2. Bloque de bruma . . . . .	10
3.8.3. Bloque normal . . . . .	10
3.9. Puntaje . . . . .	10
<b>4.</b> <i>Interfaz Gráfica e interacción</i>	<b>11</b>
4.1. Ventana de Inicio . . . . .	12
4.2. Ventana Principal . . . . .	12
4.2.1. Mostrar Tabla de clasificaciones . . . . .	12
4.2.2. Elección de DCConjunto de palabras . . . . .	13
4.2.3. Búsqueda de partida . . . . .	13
4.3. Ventana de juego . . . . .	13
4.3.1. Panel de participantes . . . . .	13
4.3.2. Editor de palabras . . . . .	14
4.3.3. Área de juego . . . . .	14
4.3.4. Otros elementos mínimos . . . . .	14
4.3.5. Bonus: Modo espectador (+4 décimas) . . . . .	15
<b>5.</b> <i>Arquitectura cliente-servidor</i>	<b>15</b>
5.1. Conexión <i>networking</i> . . . . .	16
5.2. Codificación y encriptación . . . . .	16
5.3. Servidor . . . . .	17
5.3.1. Creación de partidas . . . . .	17
5.3.2. Mecánicas de la partida . . . . .	18
5.3.3. Desconexión repentina . . . . .	19
<b>6.</b> <i>WebServices</i>	<b>19</b>
6.1. Tabla de clasificaciones . . . . .	21
<b>7.</b> <i>Base de datos</i>	<b>21</b>
<b>8.</b> <i>Archivos</i>	<b>22</b>
8.1. <i>Sprites</i> . . . . .	22
8.1.1. Bloques . . . . .	22
8.1.2. Bruma . . . . .	23
8.2. DCConjuntos de palabras . . . . .	23
8.3. <i>parametros.py</i> . . . . .	23
<b>9.</b> <i>.gitignore</i>	<b>24</b>

10.Importante: Corrección de la tarea	25
11.Restricciones y alcances	25

## 1. *DCCaída de palabras*

Tu vocación por la programación te ha impulsado a crear herramientas de aprendizaje herbóreo para la poda y cuidado de árboles. Te ha llevado a batallas legendarias en el campo de la ciberseguridad al enfrentarte contra árboles digitales malvados que deseaban conquistar todos los computadores. Y te ha abierto las puertas a la industria de la administración empresarial, optimización, y contabilidad en el comercio.

Sin embargo, en el fondo de tu corazón, siempre has sabido que lo que más deseas, es llegar a tu hogar y poder relajarte durante una tranquila tarde de invierno jugando jueguitos. Es por esto que como desafío final, te embarcarás en un proyecto que llevará a la vida este profundo deseo.

Como no pierdes oportunidad, decides que tu juego tendrá como eje principal una habilidad que ya has desarrollado con tus proyectos anteriores: la escritura a través de un teclado. Como además has ganado expertiz en trabajar arduamente durante horas, quieres que tu juego favorezca a quienes puedan resistir a lo largo del tiempo.

De aquí nace *DCCaída de palabras*, un juego que pondrá a prueba no solo tus habilidades como *gamer*, sino que también elevará al máximo tu capacidad de programar. La ronda final ha llegado; es el momento de demostrar que tienes todo lo que se necesita para transformarte en un/a desarrollador/a de videojuegos al construir este último desafío que requiere de una **Programación Avanzada**.

### 1.1. Introducción

*DCCaída de palabras* es un juego en tiempo real en que múltiples usuarios compiten por la supervivencia en una prueba de mecanografía. El objetivo principal es conservar el limitado número de vidas por la mayor cantidad de tiempo posible. Para ello, los competidores deberán escribir palabras que aparecerán en el área de juego, y que se desplazarán a lo largo de la ventana hasta llegar al borde inferior de la misma. Si el jugador no logra escribir una palabra antes de que esto ocurra, perderá una vida. El ganador de una partida es aquél jugador que logre conservar por mayor tiempo sus vidas.

Para establecer una competencia justa, *DCCaída de palabras* cuenta con una colección de [DCConjuntos de palabras](#). Estos son conjuntos preestablecidos organizados por temática que determinan las palabras que pueden aparecer durante una partida. Adicionalmente, un jugador tiene la posibilidad de crear su propio conjunto de palabras para competir en alguna temática personalizada que sea de su interés. Las reglas específicas de esto se detallan en la sección [3.2](#).

En la sección de [Flujo del programa](#) podrás encontrar una explicación de cómo debe ser la navegación en el programa, qué ventanas son accesibles desde dónde y a qué interfaces deben dirigir ciertos eventos. En la sección de [Mecánicas de Juego](#) se detallan las reglas del juego, tanto externas (según las ve y experimenta el jugador) como internas (cómo se deben manejar los elementos del juego). En [Interfaz Gráfica e interacción](#) se halla el detalle de los elementos gráficos que componen la interfaz, y qué información debe ser presentada al usuario. [Arquitectura cliente-servidor](#) aborda la conexión entre los clientes y el servidor y cómo debes preparar la información para que estos se comuniquen. La sección de [WebServices](#) detalla la estructura que debe tener el servicio de **API** que el servidor debe levantar para que los clientes y el servidor usen. Finalmente, en [Base de datos](#) se listan los archivos de datos que deberá crear y manejar tu programa.

## 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: buscar una partida, revisar la tabla de clasificaciones, consultar sus estadísticas, seleccionar uno de los [DCCconjuntos de palabras](#) ó proponer uno personalizado, 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 conjunto de palabras con el que desea participar, y ha iniciado la búsqueda de una partida, comienza un período de espera en que el servidor se encarga de buscar otros jugadores con quienes pueda competir. Este período debe ser visible para el jugador en la forma de un contador de segundos en la ventana principal. Los detalles de cómo se realiza la búsqueda y la agrupación de jugadores para una partida nueva, se detallan en [Servidor](#). Una vez acabado el tiempo de espera, se da ingreso a la siguiente ventana.

La [Ventana de juego](#) es la interfaz en que el juego en sí mismo tiene lugar. En esta ventana el jugador compite escribiendo las palabras que aparecen antes de que estas lleguen al borde del [Área de juego](#). Además, se presentan datos importantes en relación con la partida actual; nombres de los otros participantes con sus vidas restantes y puntajes actuales, tiempo total de la partida, vidas restantes del jugador, etc.

El jugador permanece en la competencia hasta que pierda todas sus vidas. Una vez que esto ocurre, existen dos posibles situaciones que se describen a continuación:

- El jugador fue **el último en perder sus vidas**: en este caso, se le debe notificar que ha ganado la partida. Con ello, se concluye la competencia, y se retorna a la ventana principal.
- El jugador **perdió sus vidas y todavía quedan participantes**: en este caso se deben presentar dos opciones:
  - Volver al menú principal: puesto que su participación ha terminado, sale de la ventana de juego y retorna a la ventana principal.
  - Permanecer en la partida actual: se transforma en un espectador de la partida en curso. Esto permite que el jugador que ha perdido continúe observando como sus oponentes sobreviven. Los detalles de esta situación se describen en la sección respectiva: [Bonus: Modo espectador \(+4 décimas\)](#).

Para ambas situaciones (el jugador ha ganado o perdido), además de lo descrito, se debe presentar información relevante de la partida relacionada con el jugador, tal como el puntaje final, el tiempo total de supervivencia, los errores cometidos, etc. Además, si ha logrado crear una nueva marca personal o global, también se le debe notificar.

### 3. Mecánicas de Juego

En esta sección se dan indicaciones específicas de cómo funciona el juego desde la perspectiva del jugador. Se define cómo participa, cómo interactúa con las mecánicas del juego, cómo puede ganar o perder, etc.

Además, se definen algunos conceptos para explicar cómo se estructura el juego y sus mecánicas, tanto internas como externas. Otros aspectos que tienen que ver con cómo se manejan las partidas y la lógica más interna del programa se describen en otras secciones como [Interfaz Gráfica e interacción](#) y [Servidor](#).

### 3.1. Tiempo de juego

La partida inicia cuando todos los participantes de ella están en la ventana de juego listos para comenzar a jugar. En ese momento, comienza un conteo de tiempo (desde cero) que lleva el registro de la duración de la partida. Este conteo continúa sin interrumpirse **bajo ninguna circunstancia** hasta que la partida termina. Cuando lo hace, el contador se detiene y el valor final corresponde a la duración total de la partida.

### 3.2. DCConjuntos de palabras

Estos son los conjuntos que el juego utiliza para determinar las palabras que pueden aparecer durante la competencia. Cada partida solo puede tener asociado un conjunto, pero pueden existir muchas partidas distintas que usen el mismo conjunto.

En principio, los conjuntos vienen preestablecidos en el programa (véase la sección de [Archivos](#)). Sin embargo, un jugador puede proponer un conjunto personalizado con una temática de su interés. Esto puede hacerlo en la [Ventana Principal](#) interactuando con los elementos correspondientes. Los conjuntos personalizados deben tener al menos [MINIMO\\_PALABRAS\\_CONJUNTO](#)<sup>1</sup> palabras distintas.

Si una partida inicia con un conjunto personalizado<sup>2</sup>, no se considerará para la tabla de clasificaciones ni para las marcas personales. Es decir, toda partida que no use un *DCConjunto de palabras* oficial (entregado por el programa y no el jugador) será únicamente para la diversión de los participantes.

### 3.3. Aparición de palabras

Las palabras que los jugadores deben ingresar se presentan por medio de apariciones en la interfaz. Cada cierto tiempo, denominado [Intervalo de aparición](#), debe mostrarse una nueva palabra. Esta aparición no está restringida por nada, es decir, es **obligatorio** que una palabra aparezca cuando se cumple el intervalo de aparición, sin importar cuantas palabras existan actualmente en el área de juego. Por lo tanto, en cualquier momento de la partida, podría existir cualquier cantidad de palabras.

La aparición de las palabras debe ser un **muestreo aleatorio sin reemplazo**<sup>3</sup>. Es decir, cada palabra que aparezca debe ser seleccionada de forma aleatoria dentro del conjunto de palabras restantes del DCConjunto de palabras que no han sido seleccionadas aún. Cuando ya se han seleccionado todas las palabras, el conjunto se vuelve a considerar en su totalidad y comienza un nuevo muestreo.

Luego de su aparición, una palabra comienza a desplazarse “hacia abajo” en el área de juego, simulando un movimiento de caída. Esto lo hará de forma continua hasta llegar al borde inferior del área. En ese caso la palabra desaparecerá y se debe consumir una vida del jugador, según lo descrito en [Supervivencia: ganar o morir](#).

La aparición de palabras es exactamente igual para todos los participantes de una partida. Es decir, cuando el juego determina que debe aparecer una palabra, a todos los jugadores debe aparecerles la misma palabra, en el mismo tiempo de juego. Además, otras propiedades de la aparición como el tiempo que tarda en caer, su posición dentro del área de juego, entre otros, quedan definidos desde el [Servidor](#) y también deben ser exactamente iguales para todos los jugadores.

---

<sup>1</sup>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](#).

<sup>2</sup>Véase [Servidor](#) para entender qué ocurre cuando hay varios jugadores que desean jugar con un conjunto personalizado.

<sup>3</sup>Muestreo sin reemplazo es un proceso en el cual los elementos se seleccionan aleatoriamente de un conjunto sin volver a incluirlos en el conjunto disponible para próximas selecciones, hasta que se agoten. Cada elección de un elemento es aleatoria dentro de los elementos no seleccionados aún. El conjunto de selecciones posibles se reduce progresivamente, por lo tanto, las elecciones posteriores dependen de las anteriores.

### 3.4. Ingreso de palabras

Como se ha mencionado anteriormente, la mecánica principal del juego es que el jugador debe **ingresar** las palabras que se le presenten. El acto de **ingresar** está compuesto de dos conceptos relacionados a las acciones del jugador

1. **Escribir una palabra:** esto consiste en el acto mismo de presionar las teclas y que el texto se muestre en la interfaz. El usuario debe poder escribir lo que desee dentro de la caja de texto de la interfaz gráfica, esto ocurre sin efectos secundarios. Esto quiere decir que el juego no considerará lo que tenga escrito hasta que realice la siguiente acción.
2. **Confirmar una palabra:** esta acción representa el deseo del jugador de enviar lo que tiene escrito en la caja de texto para que el juego la evalúe. La evaluación del juego sobre lo que el jugador escribió puede resultar en cero, una o varias explosiones de palabras.

### 3.5. Explosión de palabras

Cuando un jugador confirma una palabra, el juego debe evaluar el texto que se escribió. Si la palabra que el jugador ingresó se corresponde con una o varias palabras de las que se encuentran en el área de juego, entonces se considera una palabra correctamente ingresada, se le debe otorgar **Puntaje** y deben desaparecer del área de juego todas las apariciones de la palabra ingresada.

Si el texto que se escribió no se corresponde con ninguna palabra existente en el área de juego, entonces no hay penalización de vidas para el jugador, pero no desaparece ninguna palabra. Además un ingreso incorrecto afecta a la **Racha**

Todas las evaluaciones deben ser estrictas, es decir, sensibles a mayúsculas y caracteres especiales (como á, à, â, etc), solo se considera correctamente ingresada una palabra que coincide perfectamente con alguna de las presentes. Para simplificar, al ingreso correcto de una palabra se le denomina una **explosión** de una palabra.

#### 3.5.1. Racha

La **Racha** es un contador que siempre inicia desde cero e incrementa 1 por cada explosión que el jugador logre realizar. Sin embargo, cuando el jugador ingresa una palabra incorrecta, el contador se reinicia.

### 3.6. Supervivencia: ganar o morir

En *DCCaída de palabras*, los jugadores de una partida comienzan con una determinada cantidad de vidas, especificada por el parámetro **VIDAS\_INICIALES**, el cual debe ser mayor o igual a 1.

El objetivo del jugador es conservar sus vidas para lograr sobrevivir. Para ello, deberá explotar las palabras que aparezcan en el área de juego. Si el jugador no logra explotar una palabra antes de que esta llegue al borde inferior del área de juego, entonces se considera incompleta y el participante perderá una vida, a esto se le llama consumir una vida.

Una vez que un jugador ha consumido todas sus vidas, perderá inmediatamente la partida. El resto de jugadores presentes en la partida pueden continuar jugando. Incluso si queda un único jugador vivo, la partida continúa hasta que este consuma sus vidas, para otorgarle la posibilidad de crear nuevas marcas personales y acceder a la tabla de clasificaciones.

Por lo general, ante una **Desconexión repentina**, se debe considerar que el jugador desconectado perdió la partida. La única excepción a esto es cuando el jugador que se desconectó es el último jugador con vida, en tal caso, la partida termina y se considera a ese participante como ganador.

### 3.7. Dificultad

La dificultad es una propiedad única de cada partida que varía con el tiempo. Esta comienza con el valor **DIFICULTAD\_BASE** (mayor o igual a uno), e incrementa en **INCREMENTO\_DIFICULTAD** (mayor o igual a uno) cada **TIEMPO\_INCREMENTO\_DIFICULTAD** (mayor o igual a uno) milisegundos.

El valor de la dificultad se utiliza para determinar el **Intervalo de aparición** y el **Tiempo de caída**. Estos valores están asociados a la partida misma, por lo que se comparten entre todos los participantes.

#### 3.7.1. Intervalo de aparición

Es un valor que representa la cantidad de milisegundos que transcurren entre apariciones de nuevas palabras. Es decir, cada **intervalo\_de\_aparicion** milisegundos, debe seleccionarse una nueva palabra del conjunto asociado a la partida, y esta debe aparecer en el área de juego para que los participantes puedan explotarla.

La fórmula que se debe utilizar para calcular este valor es:

```
intervalo_de_aparicion = random.randint(  
    max(TIEMPO_APARICION_MINIMO,  
        round((TIEMPO_APARICION_MAXIMO - dificultad_actual) / 2)),  
    max(TIEMPO_APARICION_MINIMO,  
        round(TIEMPO_APARICION_MAXIMO - dificultad_actual)))
```

Donde **TIEMPO\_APARICION\_MAXIMO** es un parámetro en milisegundos que define el máximo intervalo de tiempo que puede transcurrir entre la aparición de dos palabras, a su vez, **TIEMPO\_APARICION\_MINIMO** (mayor o igual a 1) es el parámetro que establece el mínimo intervalo de tiempo en milisegundos que debe transcurrir entre la aparición de dos palabras. El valor de **TIEMPO\_APARICION\_MAXIMO** debe ser estrictamente mayor a **TIEMPO\_APARICION\_MINIMO**.

#### 3.7.2. Tiempo de caída

Es la cantidad de milisegundos que una palabra debe **desplazarse** para recorrer desde su posición de aparición en la parte superior del área de juego, hasta el borde inferior. En general, es el tiempo total que tiene el jugador para explotar la palabra antes de que esta le consuma una vida.

Notar que el tiempo de caída está ligado al desplazamiento de la palabra, y no al **Tiempo de juego**, es decir, solo se debe contabilizar el tiempo en que la palabra efectivamente **se está moviendo**. Véase **Bloque de hielo**.

Cada palabra tiene su tiempo de caída que es calculado una única vez antes de que aparezca en pantalla, en ese momento se determina y no puede cambiar. Este valor no está asociado a la palabra en sí, sino a la instancia de aparición particular. Es decir, que si en el futuro de la partida, vuelve a intentar aparecer la misma palabra, se debe calcular el tiempo de caída para esa instancia nueva y no reutilizar el calculado previamente, ni tampoco al calcular el nuevo, modificar el anterior.

La velocidad real de caída, es decir, los *pixeles/milisegundos* que viaja una palabra en la pantalla dependerá de cada instancia de cliente, y debe ser calculada para **siempre** respetar el **Tiempo de caída** calculado a partir de la dificultad<sup>4</sup>.

La fórmula que se debe utilizar para calcular el tiempo de caída:

---

<sup>4</sup>Es importante para que jugadores con una pantalla con mayor resolución no tengan ventaja respecto al tiempo que tarda en caer una palabra.

$$T_c(d) = \left\lfloor T_{max} \cdot \left( \frac{1}{1 + e^{0.08(d-50)}} \right) \right\rfloor$$

Donde

- $T_{max}$  es un parámetro [TIEMPO\\_CAIADA\\_MAXIMO](#) (mayor o igual a 300), que corresponde al máximo tiempo en milisegundos que puede tardar un bloque en caer
- $e$  es el número de Euler<sup>5</sup>.
- $d$  es la dificultad actual calculada según lo descrito en [Dificultad](#).
- Los símbolos  $\lfloor \rfloor$  indican que el resultado debe ser redondeado a un entero hacia abajo<sup>6</sup>.
- $T_c(d)$  corresponde al [tiempo\\_de\\_caida](#) para la dificultad  $d$ .

### 3.8. Bloques de palabra

Cada palabra dentro de la partida aparecerá dentro de un Bloque. Existen tres tipos de bloques en los que una palabra puede aparecer. El bloque en que aparece está determinado por la naturaleza de la palabra. Además, el tipo de bloque define el comportamiento que tendrá en relación con la partida. Al explotar una palabra, se explota el bloque en la que ella se encuentra. Cuando se define el tipo de bloque en que una palabra aparece, este no puede cambiar.

Un bloque puede tener una mecánica que beneficie directamente al jugador que le explote, ó puede entregarle ventajas indirectas al irrumpir el juego de sus contrincantes. Los bloques especiales (bloque de hielo y bloque de bruma), es decir, aquellos que añaden mecánicas al juego, aparecerán con una probabilidad determinada por un parámetro propio. Esta aparición debe ser calculada para cada participante. Esto quiere decir que una misma palabra cuya naturaleza cumpla con las condiciones necesarias para ser un bloque especial podría mostrarse como un bloque especial para un jugador, y para otro no.

Los dos tipos de bloque especial son excluyentes, es decir, una palabra no puede ser al mismo tiempo un bloque de hielo y un bloque de bruma. Si una palabra cumple todas las condiciones para ser bloque de hielo y todas las condiciones para ser bloque de bruma, deberá solo considerarse su posibilidad como bloque de hielo. Las condiciones de aparición y la mecánica asociada a cada bloque se describen a continuación.

#### 3.8.1. Bloque de hielo

Para que una palabra pueda aparecer en un bloque de hielo debe cumplir con:

- La palabra debe contener al menos un carácter no ASCII
- Al momento de aparición de la palabra, el **total de segundos** del [Tiempo de juego](#) debe ser divisible por 3.

Si una palabra cumple estas condiciones, la probabilidad de que aparezca como un bloque especial de este tipo es [PROBABILIDAD\\_BLOQUE\\_HIELO](#) (valor entre 0 y 1, inclusive).

Si el jugador explota un bloque de hielo, todas las palabras que **en ese instante** estén presentes dentro de **su área de juego**, deberán congelarse (dejar de caer) durante [TIEMPO\\_CONGELACION\\_BLOQUE\\_HIELO](#) (mayor o igual a 1) milisegundos. Una vez transcurrido este tiempo, deberán continuar su desplazamiento tal como estaban antes de detenerse.

---

<sup>5</sup>Puedes usar `math.e`

<sup>6</sup>Puedes usar `math.floor()`

Esta mecánica permite al jugador que explote un bloque de este tipo obtener más tiempo para ingresar las palabras que tiene pendientes en el área de juego y de esta forma evitar consumir vidas.

Notar que esta mecánica interactúa estrechamente con el [Tiempo de caída](#). Cuando una palabra está congelada, su desplazamiento se ha detenido, por lo que no se está contabilizando su tiempo de caída. Se debe tener cuidado de conservar el tiempo de caída restante para cada palabra. Por ejemplo, si una palabra tiene tiempo de caída equivalente a 2 segundos, pero transcurrido un segundo desde su aparición el jugador explota un bloque de hielo, entonces cuando se termine el tiempo de congelación, el tiempo que deberá tardar la palabra en llegar al borde inferior del área de juego desde su posición en ese instante será 1 segundo, que corresponde el tiempo que le faltaba por recorrer.

En caso de que un jugador explote un bloque de hielo, mientras sus palabras se encontraban congeladas por la explosión de un bloque de hielo previo, el tiempo de congelación deberá reiniciarse.

En caso de que un jugador explote una palabra que existe más de una vez en el área de juego, y existe como bloque de hielo repetidas veces, el efecto solo se aplicará una vez (pero todas las existencias explotarán).

### 3.8.2. Bloque de bruma

Para que una palabra pueda aparecer en un bloque de bruma debe cumplir con:

- Largo de la palabra mayor ó igual a [LARGO\\_MINIMO\\_BLOQUE\\_BRUMA](#).
- La suma de los caracteres ASCII debe ser un número par.

Si una palabra cumple estas condiciones, la probabilidad de que aparezca como un bloque especial de este tipo es [PROBABILIDAD\\_BLOQUE\\_BRUMA](#).

Si un jugador explota el bloque de bruma, entonces se debe seleccionar a uno de sus contrincantes de forma aleatoria. La próxima palabra que ese contrincante seleccionado reciba (solo esa), aparecerá en su área de juego parcialmente tapada (cubierta visualmente) por una Bruma que evitará ver con claridad cuál es la palabra.

La bruma debe permanecer cubriendo el bloque durante [TIEMPO\\_BLOQUEO\\_BRUMA](#) (mayor o igual a 0, menor o igual a la mitad del tiempo de caída del bloque en que aparece).

La bruma no altera la capacidad de explotar del bloque. Esto quiere decir que si el jugador afectado por una bruma logra adivinar la palabra que se encuentra escondida antes de que la bruma desaparezca, podría ingresarla y hacerla explotar.

Para el caso en que un jugador esté solo, las palabras pueden seguir apareciendo como un bloque de bruma, pero cuando este explote en ese escenario, no tendrá efecto.

### 3.8.3. Bloque normal

Este es el bloque por defecto. Por lo tanto, si una palabra no resulta en algún bloque especial, **debe** aparecer como un bloque normal.

Los bloques normales no tienen ninguna mecánica especial. Deberán ser explotados como todas las palabras.

## 3.9. Puntaje

El puntaje total obtenido durante la partida dependerá de la cantidad de palabras explotadas antes de consumir todas las vidas. Cada explosión otorga un puntaje individual calculado mediante la siguiente fórmula:

$$\text{puntaje\_explosion} = \text{largo\_palabra} \cdot \log_2(1 + \text{dificultad\_actual}) \cdot \log_2(2 + \text{racha})$$

Donde:

- **puntaje\_palabra** es el puntaje que entrega el explotar la palabra.
- **largo\_palabra** es la cantidad de **caracteres** que tiene la palabra.
- **dificultad\_actual** es el valor de la dificultad según descrito en [Dificultad](#).
- **racha** es el valor del contador [Racha](#) al momento de explotar la palabra. El cálculo de puntaje ocurre antes de que incrementar el contador.

El puntaje total del jugador, es la suma de los puntajes que obtuvo por cada explosión.

## 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 prolífico a lo largo del programa. Esto quiere decir que el usuario puede navegar sin problemas entre las distintas partes que componen *DCCaída de palabras* 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 *DCCaída de palabras* 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 explice por medio del nombre de un *widget* específico, eres libre de mostrar la información y crear la interacción por medio de los componentes que deseas, 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 aplicara 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 1 se muestra un ejemplo esquemático de los elementos mínimos solicitados con (A): nombre del programa, (B): editor de línea y (C): botón de ingreso.

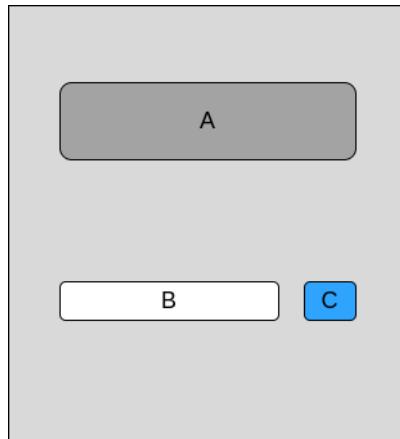


Figura 1: Esquema de ventana de inicio.

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 que 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 regresar a la ventana de inicio y mostrar un mensaje indicando la situación.

## 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.

Las acciones se describen a continuación.

### 4.2.1. Mostrar Tabla de clasificaciones

Cuando el usuario desee realizar esta acción, se deberá mostrar una tabla, lista, o formato semejante, con la información de los mejores puntajes según categoría. Queda a tu criterio si decides mostrar todas las categorías a la vez, o creas una vista que permita entregar una a elección. Esta información debe ser

obtenida desde el servidor, y puede ser desplegada dentro de la misma ventana principal, o en una ventana nueva, según lo estimes conveniente.

Puedes revisar cuáles son las tablas disponibles en la sección [Tabla de clasificaciones](#).

#### 4.2.2. Elección de DCConjunto de palabras

Esta acción puede estar asociada a un selector que despliegue los conjuntos disponibles (entregados por el servidor). Los conjuntos deben ser presentados con el nombre del mismo, la cantidad de palabras que tienen, y una breve descripción del lenguaje que contiene. Por ejemplo, la información de un conjunto se puede escribir como:

*Frutas* (144 palabras)  
“Un conjunto de todo lo comestible”

Cuando el jugador quiera utilizar un conjunto personalizado, se debe desplegar un [QFileDialog](#) que permita buscar el archivo de su conjunto personalizado. Este archivo debe seguir las reglas de los conjuntos descritas en la sección [Archivos](#). En caso de que exista un error en el formato del contenido del archivo, se debe notificar al usuario.

El conjunto seleccionado (oficial o personalizado) se puede cambiar cuantas veces se deseé mientras no se inicie la búsqueda de partida. Sin embargo, una vez que se ha iniciado, la selección debe quedar bloqueada.

#### 4.2.3. Búsqueda de partida

Una vez que se tenga seleccionado un conjunto, ya sea oficial o personalizado, se debe habilitar la acción de búsqueda de partida. Esto solicitará al servidor que cree una partida nueva donde el jugador pueda participar. Una vez iniciada la búsqueda de partida, la acción de iniciar la búsqueda se debe deshabilitar.

En la interfaz, se debe mostrar un contador que indique el tiempo de espera actual durante el cual el servidor está creando la partida. Además, debe indicarse el estado relacionado a la partida; puede ser “buscando” ó “listo para buscar”. Cuando se cree la partida, se deberá abandonar esta ventana y pasar a la siguiente etapa.

En la figura 2 se muestra una posible distribución para los elementos de la ventana principal. Con (A): la tabla de clasificaciones, (B): nombre de jugador, (C): botón para importar conjunto de palabras, (D): listado de DCConjuntos de palabras disponibles, (E): selector de conjunto , y (F): botón de buscar partida. Además de otros elementos de información como el indicador de estado de búsqueda de partida y el conteo de tiempo de búsqueda

### 4.3. Ventana de juego

Esta es la ventana donde se realiza la partida. Toda la información que el jugador necesita para participar debe presentarse en esta interfaz, por lo que es importante que maneje bien todos los elementos de los que se compone el juego.

#### 4.3.1. Panel de participantes

En este sector debe mostrarse la información relevante de los participantes. Debe existir un bloque de datos por cada jugador dentro de la partida. Cada uno de estos bloques debe contener el nombre del jugador, la cantidad de vidas restantes (sin consumir), y su puntaje actual. Toda esta información debe actualizarse en tiempo real.

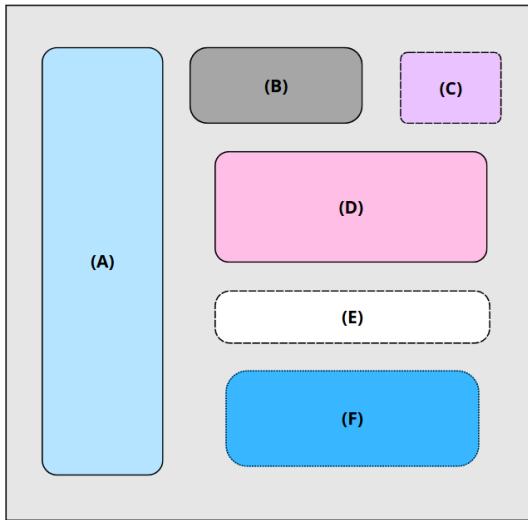


Figura 2: Esquema de Ventana Principal.

#### 4.3.2. Editor de palabras

Debe ser una forma de editor de línea que permita al jugador escribir el texto que desee para luego confirmar lo que escribió y así ingresar la palabra.

Debe reflejar perfectamente las teclas que el usuario ha presionado, es decir, si el usuario presiona teclas para escribir texto o borrar texto, se debe actualizar en tiempo real el contenido de este editor.

Además, este editor debe ser sensible a dos eventos que permiten al usuario **confirmar** lo que ha escrito: una palabra se confirma cuando el usuario presiona la **Tecla Intro** ó la **barra espaciadora**.

Cuando el jugador ingresa una palabra, el texto que contiene el editor debe borrarse por completo, de modo que el usuario pueda inmediatamente comenzar a escribir una palabra nueva.

#### 4.3.3. Área de juego

Este es el panel donde ocurre la aparición de las palabras, así como el movimiento de las mismas. Como se mencionó, las palabras deben aparecer dentro de un elemento gráfico llamado **Bloque** que indicarán la mecánica especial que tienen según lo descrito en **Bloques de palabra**.

Los bloques aparecen obligatoriamente en el borde superior de esta área de juego. La posición horizontal donde aparecen es determinada por el **Servidor**, y es la misma posición relativa para todos los jugadores. El valor entregado por el servidor es un **float** entre 0 y 1, donde 0 se corresponde con la posición “más a la izquierda posible”, es decir, la coordenada  $x = 0$  del área de juego, y 1 se corresponde con la posición “más a la derecha posible”, esta última posición es variable puesto que depende del largo del bloque particular que esté apareciendo, ya que los bloques deben siempre aparecer con la **totalidad del bloque dentro del área de juego**.

Con todo esto, es deber del cliente calcular correctamente la posición real dentro del área de juego, respecto de los valores entregado por el servidor.

#### 4.3.4. Otros elementos mínimos

Además de lo descrito, en la interfaz se debe presentar, de alguna forma que estimes adecuada, la siguiente información:

- Nombre del DCConjunto de palabras que está siendo utilizado.

- Contador que indique el [Tiempo de juego](#) actual.
- Dificultad actual, según lo descrito en [Dificultad](#).

#### 4.3.5. Bonus: Modo espectador (+4 décimas)

En esta sección se describe una funcionalidad **opcional** para tu tarea. Su correcta implementación conlleva una puntuación extra a tu calificación de esta entrega. Para poder obtener este bonus debes cumplir con los siguientes requerimientos:

- La nota en tu tarea (sin bonus) debe ser **igual o superior a 4.0**.
- El bonus debe estar implementado **en su totalidad**, es decir, **no se dará puntaje intermedio**.
- Además, deberás explicitar en tu [README](#) si implementaste esta funcionalidad.

Lo que se solicita es que cuando un jugador consume todas sus vidas y aún quedan participantes con vida, tenga la posibilidad de permanecer en la ventana de juego para observar el juego de otro participante. Cuando desea hacerlo, el [Editor de palabras](#) debe deshabilitarse y a partir de entonces, en el área de juego debe mostrarse una copia exacta del área de juego de un contrincante seleccionado al azar al momento de la muerte del jugador que está observando.

Es decir, el jugador espectador debe poder observar el juego de ese contrincante, obteniendo la información visual de las palabras que aparecen, los bloques presentes, cuáles se revientan, cuando estos se congelan, etc.

En este modo de ventana, debe presentarse un nuevo botón que permita al espectador volver a la ventana principal cuando deseé dejar de observar. Este es el único elemento de la interfaz con el que el espectador podrá interactuar mientras observe el juego.

Si el jugador a quien estaba observando consume todas sus vidas, deberá mostrarse al azar, otro contrincante con vida, de modo que el espectador siempre pueda observar la partida en curso mientras esta siga existiendo.

## 5. Arquitectura cliente-servidor

En esta sección se describe detalladamente la arquitectura que se espera utilices para desarrollar *DCCaída de palabras*. 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.

*DCCaída de palabras* 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 3. 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.

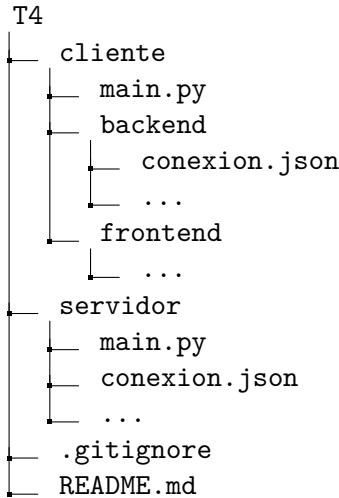


Figura 3: Organización mínima del directorio

### 5.1. 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:

```

1  {
2      "host": <dirección_ip>,
3      "puerto": <puerto>,
4  }

```

### 5.2. 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**.

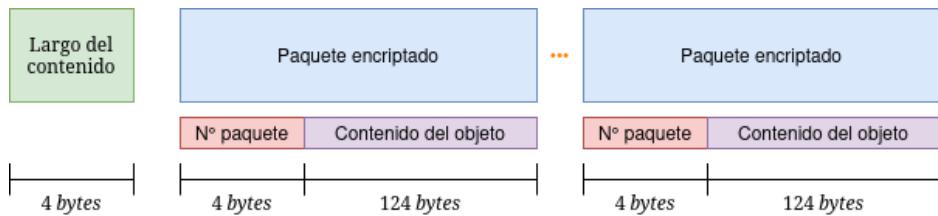


Figura 4: Ejemplo de mensaje completo

En este paso se debe **encriptar** cada paquete. Para ello se deberá realizar una operación **XOR** por cada *byte* del mensaje utilizando una clave de 128 *bytes* llamada **CLAVE\_CIFRADO**. El listado de paquetes debe ser desordenado aleatoriamente antes de ser enviado<sup>7</sup>.

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\_CIFRADO**. En la figura 4 se puede observar un diagrama que ilustra cómo se debe componer el mensaje enviado.

### 5.3. 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, la recepción y verificación de las palabras ingresadas por el jugador, y toda acción relevante respecto al flujo del juego y la producción de sus mecánicas.

#### 5.3.1. Creación de partidas

La primera labor del servidor para el manejo del juego es la agrupación de jugadores para la creación de partidas. Un cliente solo puede entrar a una **partida nueva**, en ningún caso un cliente se podrá conectar a una partida que ya se encuentra en curso.

La agrupación de jugadores es la técnica para definir los participantes de una partida nueva, esta depende de dos factores: el **tiempo de búsqueda de partida**, y el DCConjunto de palabras utilizado en la partida. Para manejar correctamente la creación se introduce el concepto de **grupo de espera**.

Un grupo de espera corresponde a un conjunto de clientes que cumplen con haber seleccionado un mismo DCConjunto de palabras y estar buscando partida (ambas cosas indicadas por el cliente mismo cuando el usuario interactúa con los elementos de la interfaz gráfica respectivos).

Cuando el cliente comienza a buscar partida, el servidor debe buscar entre los grupos de espera existentes, si hay alguno que tenga asociado el mismo DCConjunto de palabras para integrar al nuevo cliente en ese grupo.

Si no existe un grupo de espera que tenga asociado el DCConjunto de palabras que ese cliente seleccionó, entonces el servidor crea un nuevo grupo de espera a partir de ese cliente (es decir, con ese cliente como el primer participante).

Un grupo de espera creado tendrá asociada una cuenta regresiva que inicia inmediatamente. Esta cuenta tiene una duración de **TIEMPO\_MAXIMO\_BUSQUEDA\_PARTIDA** (mayor o igual a 1) segundos. Durante estos segundos el servidor esperará a otros clientes que deseen iniciar una partida con el mismo DCConjunto de palabras. Si estos clientes existen, los integrará al grupo de espera actual y permanecerán en él hasta que se cumpla el tiempo de la cuenta regresiva.

<sup>7</sup>Puedes usar `random.shuffle()`

Una vez terminada la cuenta regresiva, el servidor crea y da inicio a la partida con el DCConjunto asociado al grupo. Los participantes son todos los clientes pertenecientes a ese grupo. Una vez que la partida inicia, el grupo deja de existir, por lo que si un nuevo cliente desea jugar utilizando el mismo DCConjunto, deberá crearse un nuevo grupo e iniciar una nueva cuenta regresiva.

Todo esto implica que el tiempo real que un cliente espera para iniciar una partida dependerá de si otros clientes ya están esperando el inicio de una partida con el mismo DCConjunto seleccionado. En el peor de los casos, el cliente esperará la totalidad del tiempo máximo, y entrará a una partida como único participante.

**Conjunto personalizado de palabras** Dado que los usuarios pueden utilizar conjuntos personalizados para crear una partida utilizando palabras de su elección, será necesaria la administración de un grupo de espera especial, denominado grupo de espera personalizado.

Este grupo busca juntar todos los usuarios que deseen jugar con un conjunto personalizado de palabras. Lo que debe hacer el servidor en este caso, es recibir el conjunto personalizado de cada uno de los jugadores, unirlos y crear un conjunto final que tendrá de nombre “personalizado”.

Ese conjunto, mezcla de todos los enviados por los clientes, es el que será utilizado para la partida del grupo de espera personalizado.

Un jugador que seleccione un conjunto personalizado, obligatoriamente será parte de este grupo de espera personalizado. Solo puede existir un grupo de espera personalizado a la vez, cada usuario que desee iniciar partida con su conjunto no oficial, formará parte de este grupo.

La partida asociada al grupo de espera personalizado no cuenta para las tablas de clasificación.

### 5.3.2. 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 y la explosión de palabras. Entre ellos, se incluyen:

- El contador con el [Tiempo de juego](#).
- El registro de la [Dificultad](#) actual.
- El conjunto completo de las palabras que están asociadas a la partida.
- La evaluación de los [ingresos](#) de palabra que haga el jugador.
- El registro de [Racha](#) de cada jugador.
- Las **vidas** de cada jugador.
- La dificultad actual.
- El intervalo de aparición, y la aparición de la palabra.
- El tiempo de caída.
- La posición horizontal donde debe aparecer cada palabra seleccionada.
- La selección de palabras del conjunto según el muestreo descrito en [Aparición de palabras](#).
- La determinación del bloque en que cada palabra aparece.
- El puntaje de cada jugador.

- La determinación de jugador afectado por bruma y si el bloque tiene bruma.

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 interactuar con la **API** descrita en *WebServices* para actualizar las tablas de clasificación, y actualizar los datos de usuarios.

### 5.3.3. 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 como que perdió todas sus vidas y será eliminado de la lista de jugadores activos. La partida continuará con los jugadores que permanezcan en línea, ó se terminará si el jugador desconectado era el último en pie.

## 6. *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 *public*, mientras que otros requieren el uso de *public* autenticación para validar la acción asociada a él *private*. Para la autenticación, 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 en el archivo `.json` que se describe en *Conexión networking*. Deberá estar disponible en todo momento mientras el servidor se esté ejecutando. **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()`: `cls`, `object_hook`, 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`, ajustando los valores de `response` ó `status` según corresponda.

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 ellos.

**(public) GET /rankings** Debe obtener la tabla de clasificación solicitada a través de los parámetros.

- Parámetros:

- **nombre**: es el nombre de la tabla de clasificaciones a la que se desea acceder, los nombres se enumeran en [Tabla de clasificaciones](#).
- **cantidad**: es un **int** para indicar el número de entradas de la tabla que se desean obtener. Si no se especifica, debe ser **MINIMO\_ENTRADAS\_CLASIFICACION**.
- **conjunto**: es el nombre del conjunto que se busca consultar. Esto se utiliza para las tablas de **Supervivencia-lenguaje** y **Puntaje-lenguaje** (véase [Tabla de clasificaciones](#)).

⇒ **Retorna:** Un JSON con los mejores jugadores de la tabla.

---

**(public) GET /conjuntos** Debe empaquetar y retornar en un JSON la información de los DCConjuntos de palabras. Solo debe incluir la información de los conjuntos y no las palabras, es decir, por cada conjunto debe incluir el **nombre**, **descripción** y la **cantidad de palabras**.

⇒ **Retorna:** Un JSON con la información de todos los conjuntos.

---

**(private) POST /users** Este *endpoint* es utilizado por el servidor cuando un nuevo jugador ha ingresado por primera vez. Agrega al archivo de usuarios el nombre del jugador y le asigna el estado de conectado.

**Header:** debe incluir el *token* de autenticación para que solo el servidor pueda acceder.

**Parámetros:** debe incluir el **nombre de jugador** del nuevo usuario conectado.

⇒ **Retorna:** Un código de éxito o error según corresponda.

---

**(private) GET /users** Este *endpoint* es utilizado por el servidor para consultar por un usuario específico. Si existe, deberá retornar el estado de conexión de ese usuario. Si no existe, deberá retornar un mensaje de error coherente.

**Header:** debe incluir el *token* de autenticación para que solo el servidor pueda acceder.

**Parámetros:** debe incluir el **nombre de jugador** por el que se consulta.

⇒ **Retorna:** Error si el usuario no existe. El estado de conexión en caso de que sí exista.

---

**(private) PATCH /users** Este *endpoint* es utilizado por el servidor para actualizar el estado de conexión de un usuario existente. Debe modificar el archivo de usuarios para asignar el nuevo valor.

**Header:** debe incluir el *token* de autenticación para que solo el servidor pueda acceder.

**Parámetros:** debe incluir el **nombre de jugador** y el **estado de conexión** que se desea actualizar.

⇒ **Retorna:** Un código de éxito o error según corresponda.

---

**(private) POST /games** Este *endpoint* es utilizado por el servidor cuando una partida ha terminado. Agrega al archivo de partidas toda la información de la partida que se debe guardar. Esto es el **identificador**, los **jugadores** que participaron, el **tiempo de supervivencia** de cada uno, y el **puntaje** de cada uno.

**Header:** debe incluir el *token* de autenticación para que solo el servidor pueda acceder.

**Body:** debe ser el diccionario con el formato de una partida según se observa en [Base de datos](#).

⇒ **Retorna:** Un código de éxito o error según corresponda.

## 6.1. Tabla de clasificaciones

Las tablas de clasificaciones son listados ordenados que contienen los mejores jugadores según distintos criterios. Para obtenerlos, la parte del servidor que maneja la **API** debe interactuar con los archivos de la [Base de datos](#) y determinar los nombres de quienes pertenecen a la lista, y su posición dentro de ella.

Las tablas de clasificaciones mínimas que deben existir son:

1. **Supervivencia:** Esta tabla ordena los jugadores con los mejores tiempos de supervivencia, es decir, los jugadores que hayan sobrevivido más tiempo jugando independiente de su puntaje o el DCConjunto de palabras utilizado. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **tiempo de supervivencia**.
2. **Supervivencia-lenguaje:** Esta tabla ordena los jugadores con los mejores tiempos de supervivencia asociados a partidas que utilizan un DCConjunto de palabras específico. Sin importar el puntaje obtenido. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **tiempo de supervivencia**.
3. **Puntaje:** Esta tabla ordena según los mejores puntajes obtenidos en las partidas. Es decir, debe buscar los puntajes más altos que se han obtenido en todas las partidas sin importar el DCConjunto de palabras ni el tiempo de supervivencia. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **puntaje final**.
4. **Puntaje-lenguaje:** Esta tabla ordena según los mejores puntajes obtenidos en las partidas asociadas a un DCConjunto particular. Es decir, busca los puntajes más altos que se han obtenido en todas las partidas que utilizaron ese conjunto, sin importar el tiempo de supervivencia. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **puntaje final**.
5. **Adicción:** Esta tabla ordena los jugadores respecto a la cantidad de partidas que han jugado, entregando los mejores puestos a los usuarios que han participado en la mayor cantidad de partidas, sin importar si ganaron o perdieron. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **partidas jugadas**.
6. **Victorias:** Esta tabla ordena los jugadores respecto a la cantidad de victorias que tienen. La información de cada puesto de la tabla debe contener **nombre del jugador**, y **cantidad de partidas ganadas**.

## 7. *Base de datos*

En esta sección se describen los archivos que se deben utilizar para registrar toda la información del juego. Estos archivos son leídos y modificados a través de los [WebServices](#) que ofrece el servidor, se interactúa con ellos únicamente a través de la **API**.

Los archivos mínimos deben ser:

1. **usuarios.csv:** en cada línea de este archivo se encuentra la información del nombre de un usuario, y si este se encuentra en línea o no. Por ejemplo:

1	PedroPicapiedra, False
2	Jhonny Depp, True

```
3 Jhonny Bravo, True  
4 rogueMaster, False
```

2. `partidas.csv`: en cada línea de este archivo se guarda la información del *id* de la partida, la duración total, el nombre del DCConjunto de palabras utilizado (“personalizado” para los conjuntos personalizados), y el nombre del jugador que ganó la partida. Por ejemplo:

```
1 0,00:15:22,Frutas,WinniehThePoh  
2 1,00:22:01,Nombres,rogueMaster  
3 2,00:01:40,personalizado,Jhonny Depp  
4 3,01:26:41,Pokemons,otakun
```

3. `partidas_usuarios.json`: debe contener la información de todas las partidas. Debe seguir la siguiente estructura:

```
1 [  
2 {  
3     "id_partida": <id_partida>,  
4     "usuarios": [  
5         {  
6             "nombre": <nombre_jugador>,  
7             "supervivencia": <tiempo_supervivencia>,  
8             "puntaje": <puntaje_final>  
9         },  
10         ...  
11     ]  
12 },  
13 ...  
14 ]
```

## 8. 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.

### 8.1. Sprites

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

#### 8.1.1. Bloques

Esta es la representación de los [Bloques de palabra](#) en que cada aparición de palabra debe ocurrir.



(a) Bloque normal



(b) Bloque de hielo



(c) Bloque de bruma

Figura 5: Bloques del juego

### Avanzada

(a) Bloque normal con palabra

### Alineamiento

(b) Bloque de hielo con palabra

### Enciclopedia

(c) Bloque de bruma con palabra

Figura 6: Bloques del juego con palabras

#### 8.1.2. Bruma

La bruma es el elemento que cubre a los bloques de los contrincantes cuando un jugador explota un bloque de bruma. Debe aparecer por sobre el bloque y la palabra, cubriendole para entorpecer la lectura.



(a) Bruma



(b) Un bloque normal cubierto con bruma

Figura 7: Bruma y palabra afectada con bruma

## 8.2. DCConjuntos de palabras

Estos archivos contienen la información de cada conjunto de palabras. La primera línea del archivo contiene la descripción del conjunto. A partir de la segunda línea se listan todas las palabras que pertenecen al conjunto. Solo el servidor debe tener acceso a los DCConjuntos de palabras oficiales, el cliente no debe tener una copia de estos recursos. Por ejemplo, un archivo DCConjunto de palabras llamado `verduras.txt`:

1	Cositas vegetales para hacer comida
2	Tomate
3	Manzana
4	Papa
5	Apio

## 8.3. `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:

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

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

```
1 PI = 3.14  
2 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**<sup>8</sup>, ya que esta práctica se considera incorrecta y su uso conllevará una reducción en tu calificación.

## 9. .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 (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.

---

<sup>8</sup>*Hard-coding* es la práctica de ingresar valores directamente en el código fuente del programa en lugar de parametrizar desde fuentes externas.

## 10. 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 señalara 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 23 de junio 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 25 de junio 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](#).

## 11. 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 3.11.X con X mayor o igual a 7.
- 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 carpeta 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).**