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

**IIC2233 Programación Avanzada (2018-2)**

# Tarea 04

## Entrega

- **Entregable (obligatorio)**
  - **Fecha y hora:** domingo 11 de noviembre de 2018, 23:59
  - **Lugar:** Cuestionario de Google Docs
- **Tarea**
  - **Fecha y hora:** domingo 25 de noviembre de 2018, 23:59
  - **Lugar:** GitHub — Carpeta: `Tareas/T04/`
- `README.md`
  - **Fecha y hora:** lunes 26 de noviembre de 2018, 23:59
  - **Lugar:** GitHub — Carpeta: `Tareas/T04/`

## Objetivos

- Aplicar conceptos y nociones de *networking* para la creación de un servicio en línea
- Diseñar e implementar protocolos para el envío de mensajes entre cliente y servidor
- Aplicar conceptos del serialización para el manejo de envío de archivos
- Aplicar conocimientos de diseño y creación de interfaces gráficas
- Aplicar conocimientos de *threading*

# Índice

# 1. Introducción

Tras las innumerables ganancias obtenidas en el mundo de los negocios, *il Tini* decide expandir su imperio al mundo de los juegos y realizar una versión de uno de los clásicos de su infancia en la isla de Sicilia. Como los negocios no andan muy bien para utilizar *engines* de videojuegos[1], te encargó la misión de desarrollar por completo un juego que, con *animaciones fabulosas*, desbanque a *Progranite* del estrellato. Mediante tus avanzados conocimientos de PyQt5, *threading* y *networking*, deberás programar *DCCurve*.

# 2. *DCCurve*

*DCCurve* es un juego multijugador con estilo *last man standing*. Aterrizado a este contexto, los jugadores tendrán que competir entre ellos, de manera simultánea, con el objetivo de mantenerse como el último jugador sobreviviente en un terreno rectangular. En este mapa, los participantes, cada uno con un color distinto del resto, se desplazarán siempre en el mismo sentido. Mientras hacen esto, dejarán una línea indeleble en el mapa —del color asignado al jugador— como rastro del camino que han seguido. Asimismo, es importante notar que los jugadores no podrán dejar de avanzar; sólo pueden controlar la dirección de su trayectoria. Además, cada cierto instante aleatorio y de forma involuntaria, los jugadores dejarán de *pintar* su rastro por una pequeña fracción (también aleatoria) de tiempo. Para sobrevivir, los competidores deberán utilizar sus habilidades para evitar una colisión tanto con el borde del mapa como con algún rastro, sin importar si es propio o perteneciente a un rival. Si todavía tienes dudas, puedes hacer clic **aquí** para observar un ejemplo clarificador.

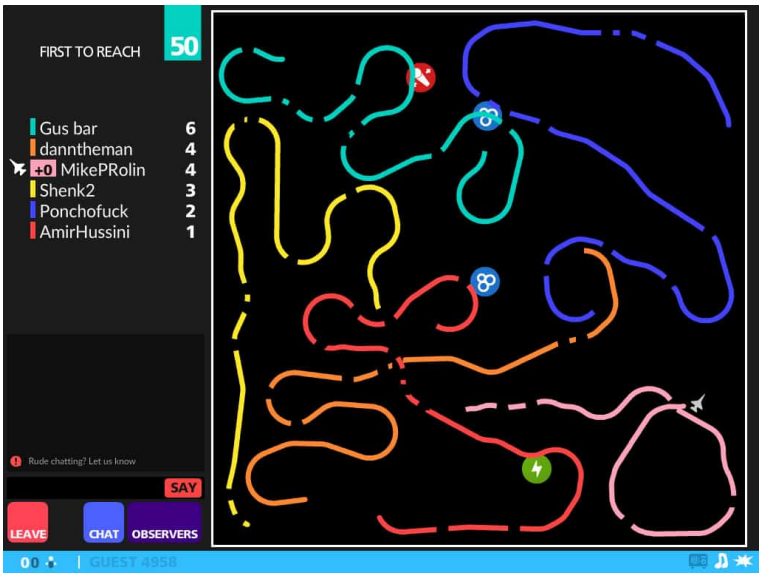


Figura 1: Un ejemplo de una posible implementación del juego

## 2.1. Repartición de puntaje

No es posible hablar de competencia sin un sistema de puntajes. Para implementar este sistema en *DCCurve*, cada partida está dividida en rondas sucesivas en donde los jugadores pueden obtener puntos. A medida que los jugadores quedan eliminados de una ronda, deberán esperar hasta que quede sólo un jugador vivo para así comenzar la siguiente ronda. Además, cada vez que un jugador queda eliminado,

[1]Para más información, puedes visitar `https://es.wikipedia.org/wiki/Motor_de_videojuego`.

el juego asignará un punto a todos los jugadores que siguen en pie, y así sucesivamente hasta que quede un único sobreviviente. En ese momento, termina la ronda y comenzará una siguiente. Estas asignaciones de puntaje se materializan únicamente al final de cada ronda. De esta manera, si tenemos una ronda con $n$ jugadores, cada una de ellas terminará con una asignación en el que existe un único jugador que no obtuvo ningún punto, otro que obtuvo un punto... y así hasta tener un jugador que obtuvo $n-1$ puntos. Sin embargo, existe una excepción a esta regla: en una ronda podrían haber dos (o más) jugadores que no pierden de forma simultánea. Este caso será explicado en la siguiente subsección.

El puntaje se va acumulando entre rondas y sumándose según fue descrito previamente. La cantidad de rondas es indefinida, y el juego seguirá hasta que se cumplan las siguientes condiciones: **a)** que algún jugador llegue al menos a $X$ puntos, donde $X$ es definido por el «jefe de la partida» (según lo indicado en la sección 4.2) y **b)** que exista una diferencia igual o mayor a dos puntos entre el jugador en primer y segundo lugar. Es importante notar que esta última regla imposibilita que existan empates. Naturalmente, el jugador que obtenga mayor puntaje será el ganador de la partida.

### 2.1.1. Eliminación simultánea

Lo descrito previamente sólo funciona cuando los jugadores pierden en momentos distintos de la ronda. Sin embargo, podría ocurrir que dos jugadores[2] se eliminan al mismo tiempo. Esto sucede específicamente cuando uno choca con el otro de frente. Para este caso, se considerará que el jugador con mayor puntaje perdió primero. En caso de que estén empatados en puntaje, tendrá ventaja el que tenga más *nebcoins*[3] (ver sección 2.5). Por ejemplo, dada una partida con el siguiente *score*,

```
Alice, 3
Bob, 4
Charlie, 5
```

Ahora, supongamos que Alice y Bob chocan entre ellos de frente, eliminándose al mismo tiempo. A partir de lo descrito anteriormente, los puntajes quedarían así,

```
Alice, 4
Bob, 4
Charlie, 6
```

## 2.2. Inicialización de la partida y rondas

Todos quieren empezar la partida cuanto antes, pero... ¿cómo piensan moverse? Antes de iniciar la partida, cada jugador debe primero elegir las teclas que utilizará para moverse. Deben ser dos: una para girar a la izquierda y otra para la derecha. Una vez que un jugador esté listo, debe esperar hasta que todos los participantes restantes estén preparados para comenzar.

Al iniciar la ronda a cada jugador se le debe asignar una ubicación dentro del mapa y una dirección aleatoria. Esta será el estado inicial del jugador. Como no hay gloria en ganarle a un oponente que murió inmediatamente después de iniciar el juego por una mal estado inicial, deberás preocuparte de que estos estados no ocurran. En la figura 2 se presentan algunos casos de malos estados iniciales. En estos, un jugador morirá casi inmediatamente luego de iniciar el juego.

Una vez que todos los jugadores estén ubicados en el mapa de juego, se debe esperar a que alguno de los jugadores presione el botón de la barra espaciadora para iniciar el movimiento.

[2] Bueno, también podría ocurrir con más de dos jugadores, pero sería bastante improbable.

[3] «¿Y si están empatados en *nebcoins*?»—preguntó un alumno. Entonces queda a tu criterio cómo desempatar.

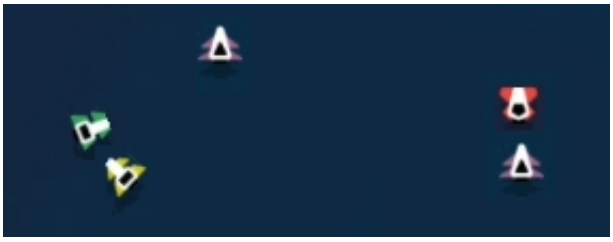

Figura 2: Ejemplo de algunas «malas ubicaciones»

### 2.3. Movimiento continuo

Como se mencionó previamente, el movimiento es constante hacia adelante. Para maniobrar y esquivar las trampas en el campo de juego, cada jugador puede controlar su movimiento por medio del teclado. En esta sección se hablará de «teclas de dirección» como las que tiene cualquier teclado, pero recuerda que cada jugador puede elegir las propias al iniciar la partida.

Utilizando las teclas de dirección, se le dará movimiento hacia la izquierda o derecha del jugador. Este eje de referencia es relativo a dónde se apunta el avance en el juego. Por ejemplo, en la figura 5 se representa la dirección de avance con la punta de una flecha, y presionar la izquierda o la derecha hará que este cambie su dirección independientemente de si el avance es hacia la zona inferior de la pantalla o no.


Figura 3: Teclas de dirección habilitadas para el movimiento con entrada teclado

Sin embargo, el ángulo de giro no es de 90°. Este se rige según una circunferencia de radio $\rho$ desde el personaje, el cual permite un movimiento de rotación al cambiar de dirección en vez de ir describiendo líneas rectas. Los cambios de dirección no deben ser inmediatos, por lo que el ángulo de giro no debe ser de 90°. En vez, los jugadores irán describiendo curvas a través del mapa tal y como se muestra en las figuras 4 y 5


Figura 4: Ejemplificación de giro hacia la derecha manteniendo pulsada tecla de dirección

Estas curvas podrán ir tomando diferentes formas según la frecuencia con la que se presionen las teclas de dirección escogidas.

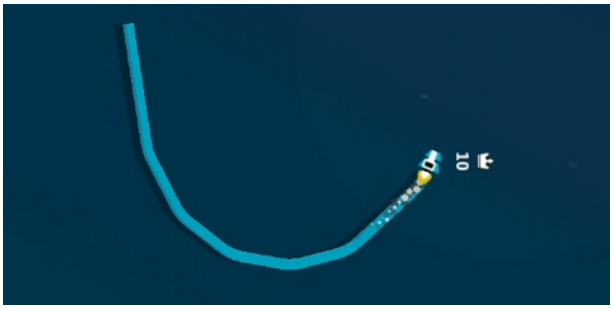

Figura 5: Ejemplo de curva formada con distintos ángulos de giro

## 2.4. Un rastro peligroso

A medida que un jugador va moviéndose por el mapa irá dejando un rastro de su color asignado. Estos rastros que van dejando los distintos jugadores harán de barrera en el mapa, y si **cualquiera** de los jugadores colisionan con estos, entonces morirán inmediatamente. Sin embargo, para permitir una mayor maniobrabilidad de los jugadores, cada cierto tiempo aleatorio $T$ segundos, donde $T$ distribuye uniforme continua $U(a = 0, b = \tau)$, se corta el rastro durante $S$ segundos y $S$ distribuye uniforme continua $U(a = 0, b = \delta)$. Debes definir $\delta$ y $\tau$ de manera que permita una buena jugabilidad.

Cuando un jugador muere en una ronda, su rastro debe mantenerse de manera que siga dificultando el libre movimiento de sus contrincantes.

Figura 6: Un *zoom* de ronda en proceso, con jugadores eliminados

## 2.5. Poderes

Para que el juego sea más dinámico existen distintos poderes, que alteran el juego momentáneamente. Estos irán apareciendo aleatoriamente en algún lugar del terreno, cada $X$ segundos, donde $X$ distribuye uniforme $U(a = 5, b = 10)$. Cada vez que aparezca un poder se deberá seleccionar, de manera equiprobable, uno del conjunto de poderes disponibles para ese juego según eligió el «jefe de la partida» (según lo indicado en la sección 4.2).

Cada poder se verá representado por un ícono, el que aparecerá en el terreno de juego. La elección de cada ícono es libre; sin embargo, si estás sin ideas puedes usar algún *emoji*, como se muestra en la figura 7. Al momento de colisionar con uno de estos poderes, el poder debe desaparecer del juego luego de aplicar su

respectivo efecto. Finalmente, si un poder no es activado después de seis segundos de su aparición, este debe ser removido del mapa.

Existen seis poderes. Estos se detallan a continuación.

- **Usain Nebolt:** Al momento de obtener este poder, el jugador aumenta su rapidez de movimiento al doble de su rapidez actual. Luego de cinco segundos, el jugador reduce su velocidad en la mitad (*i.e.* volviendo a su velocidad original). Además, los aumentos de velocidad son acumulables.
- **Fernando Limpiessa:** Una limpieza en el mapa. Al momento que cualquier jugador obtenga este poder, los rastros de todos los jugadores son eliminados inmediatamente del mapa. No obstante, los poderes que estén en el mapa no son eliminados.
- **Jaime Sinrastro:** El jugador dejará de marcar su rastro y además ignorará todo tipo de colisiones por cuatro segundos; es decir, durante ese periodo, el jugador no podrá obtener otros poderes y no podrá morir por choques con otros rastros, aunque sí morirá si choca con el margen del mapa.
- **Fernando Cervessa:** Los efectos del alcohol. Las teclas del jugador invierten su sentido: con la tecla que solía ir hacia la derecha ahora se moverá a la izquierda y viceversa. Este poder dura cinco segundos.
- **Felipe del Trío:** Al momento que un jugador obtenga este poder, deben aparecer **tres** poderes al azar (esto también incluye a *Multipoder*) en el mapa.
- **Nebcoins:** Estas monedas sólo sirven para desempatar en caso de un choque frontal. Y ojo, esto es sólo un uso de los cientos de miles que tienen las ***nebcoins*** en la vida real.

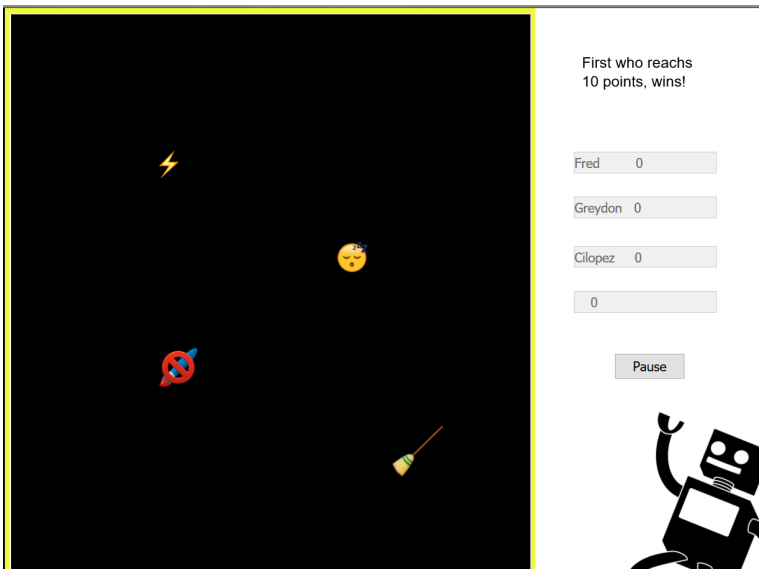


Figura 7: Un ejemplo de un posible mapa con cuatro poderes

## 3. Funcionalidades

Para hacer más cómodo el trabajo del cliente, toda interacción con este debe realizarse a través de una interfaz gráfica. Esta debe ser amigable y enfocada en la usabilidad; es decir, comportarse de acuerdo a lo que espera el usuario de forma natural, sin eventos inesperados.

### 3.1. Sistema de autentificación (registro e ingreso de usuarios)

Al iniciar el programa, este deberá ofrecer dos opciones: registrarse si es un usuario nuevo, o ingresar si es que ya está registrado.

#### 3.1.1. Registro de usuarios

Si es que se escoge la opción de registro, el programa debería pedir un nombre de usuario, contraseña y confirmación de la contraseña. Se debe verificar que el nombre de usuario sea único entre todos los usuarios de *DCCurve*, y que la contraseña y la verificación de la contraseña sean iguales. La base de datos de los usuarios debe persistir. Dicho de otra forma, los usuarios registrados durante una ejecución del programa deberán estar disponibles en una ejecución en un momento diferente. Para hacer esto, queda a su criterio el mecanismo a implementar.

Existe un requerimiento especial de seguridad del sistema: las contraseñas **nunca deben quedar guardadas en el disco duro sin encriptar** (*i.e.* en texto plano). Para encriptarlas, se debe usar un protocolo de ***hash + salt***[4] que se detalla a continuación: Cuando se escriba un nuevo usuario en el disco duro debes,

1. Generar una secuencia de *bytes* aleatoria de largo 8. Esta secuencia será nuestra «sal». Se recomienda fuertemente que la obtengas a través del método `urandom` del módulo `os`.
2. Concatenar la sal con la codificación en *bytes* de la contraseña del nuevo usuario. Recuerda que puedes obtener la codificación en *bytes* a través del método `encode` de un *string*. La concatenación de ambas secuencias será nuestra secuencia a encriptar.
3. Pasar la secuencia a encriptar por alguna función de *hash*. Se recomienda fuertemente usar el algoritmo `sha256` a través del módulo `hashlib`. La secuencia de *bytes* que sea retornada por la función de *hash* será nuestra contraseña encriptada.

#### 3.1.2. Ingreso de usuarios

Si el usuario selecciona la opción de ingresar, se le debe pedir su nombre de usuario y contraseña. Debes verificar que el nombre de usuario exista y que la contraseña para dicho usuario sea la correcta. Sin embargo, como la contraseña estará encriptada en la base de datos debes verificar que sea correcta de la siguiente manera:

1. Obtener desde la base de datos de *DCCurve* la sal y contraseña encriptada del usuario que está intentando ingresar.
2. Concatenar la sal con la secuencia de *bytes* de la codificación de la contraseña no encriptada; es decir, la contraseña que el usuario que está intentando ingresar escribió. Esta será la secuencia a verificar.
3. Pasar la secuencia a verificar por la misma función de *hash* que se utilizó al momento de registrar al usuario. La secuencia de *bytes* que retorna de la función de *hash* corresponderá a la secuencia encriptada a verificar.
4. Realizar la comparación entre la secuencia encriptada a verificar y la contraseña encriptada. Si son iguales, entonces la contraseña que ingresó el usuario es correcta. De lo contrario, la contraseña es incorrecta.

[4] Puedes profundizar de por qué se utiliza este protocolo **aquí**.

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

### 3.2. Sala de espera

Una vez iniciada la sesión, el servidor asignará al usuario a alguna partida que se encuentre por comenzar. En el caso de que no hayan partidas disponibles se abrirá una. Las partidas constan de una sala de espera donde los jugadores entrarán por orden de llegada, donde el primero se clasificará como «jefe de partida» y el resto de los jugadores se irán acumulando hasta llegar a un máximo de 4. En el caso de que el jefe de partida abandone la partida será reemplazado por el jugador de más antigüedad en la sala. Una vez comenzada una partida, sin importar si faltan jugadores o no, cualquier otro jugador que busque partida creará una nueva sala de espera. Además, esta sala de espera contiene un chat, donde los usuarios pueden interactuar mientras esperan a que su partida inicie. En conjunto con este esquema, la sala de espera se debe organizar de la forma más clara posible para proveer de la mejor experiencia posible a sus usuarios. Los requisitos mínimos están explicados en la sección 4.2.

## 4. Interfaz gráfica

Esta sección tiene por objetivo enunciar lo mínimo que deberá tener cada ventana de tu interfaz gráfica.

### 4.1. Ventana de inicio

En esta ventana deben haber dos botones; uno para ingresar al juego y otro para registrarse en el juego. Dependiendo del botón presionado, se abren dos ventanas (o se modifica la ventana de inicio) para que aparezca lo siguiente:

- Si se presiona **Ingresar**: Tienen que aparecer dos cuadros de texto, uno para ingresar usuario y otro para ingresar contraseña, y un botón para ingresar los datos. Si los datos no existen, debe aparecer un mensaje señalándolo.
- Si se presiona **Registrar**: Lo mismo que para ingresar, más un cuadro de texto para confirmar la contraseña. Si el usuario ya existe, tiene que aparecer un mensaje avisando la situación.

Para ambas ventanas, debe haber un botón para volver a la ventana anterior, porque el usuario pudo haberse equivocado y apretado «Ingresar» en vez de «Registrar», o viceversa. En caso de que se haya modificado la misma ventana de inicio, se debe poder seguir presionando el botón contrario al que se apretó antes.

### 4.2. Sala de espera

Hay dos tipos de sala de espera, uno para el jefe de partida y otro para el resto de los jugadores. Se debe cumplir lo siguiente:

- Para ambos tipos de sala de espera, deben aparecer las siguientes funcionalidades:
  - **Lista de participantes:** Debe existir una visualización de los jugadores que van ingresando a la partida, con un máximo de 4, ordenados por orden de llegada, con su nombre de usuario y un color entregado por *default*, siendo todos los colores distintos. Se debe indicar explícitamente cuál jugador es el jefe de partida (el que entra primero a la sala de espera).

- **Selección de color:** Cada jugador debe tener la opción de cambiar su color, sin poder tener el mismo que otro. Esta selección se puede hacer con una lista desplegable.
- **Chat:** No hay mejor forma de pasar el tiempo en una sala de espera que hablando con tus amigos. Por esto mismo, se deberá implementar un chat en el que los participantes puedan escribir mensajes que todos en la sala puedan ver. Los mensajes deben estar por orden cronológico, de más antiguo a más reciente, indicándose en cada mensaje el nombre del usuario que mandó el mensaje. En el chat, debe haber un cuadro de texto donde se pueda ingresar un mensaje y un botón para enviarlo.
- **Cuenta regresiva:** Una vez que el jefe de partida presione el botón de «Inicio de partida», deberá aparecer un temporizador haciendo una cuenta regresiva de 10 segundos para partir el juego. Una vez que llegue a cero, se dará inicio a la primera ronda del juego.

- Además, la sala de espera del jefe de partida, debe tener las opciones descritas a continuación, y puedes permitir que los jugadores que no son jefes de partida, vean las opciones que este ingresó, pero no modificarlas.
  - **Selección de parámetros:** El jefe de partida deberá poder seleccionar los poderes que van a aparecer en la partida. Queda a tu criterio cómo implementar esto: con una lista desplegable de selección múltiple, usando *checkboxes* por poder, etcétera. Además, se debe poder seleccionar la velocidad del juego y el puntaje necesario para llegar a la victoria.
  - **Botón de inicio de partida:** Finalmente, debe aparecer un botón para dar inicio a la partida, el que activará el cronómetro de la cuenta regresiva. Este botón sólo se puede apretar si hay mínimo dos jugadores en la sala de espera (incluyendo al jefe de partida).

Además, debe haber un botón para salir de la sala de espera (volver a la ventana de inicio). Pero cuando empiece la cuenta regresiva, este botón debe bloquearse. Una vez que termine la cuenta regresiva, se debe pasar a la sala de juego.

### 4.3. Sala de juego

Esta ventana debe tener:

- **Zona de juego:** Un cuadrado que delimite el área donde se realiza el juego. Debe ser lo suficientemente grande, como para que el desarrollo del juego se entienda con claridad. Se recomienda utilizar un fondo negro.
- **Zona de datos:** A cualquiera de los lados de la zona de juego, debe haber una zona donde se indique lo siguiente:
  1. El listado de los jugadores, con su color, nombre de usuario y puntaje.
  2. El puntaje necesario para la victoria.
  3. Un botón para pausar el juego.
  4. Un botón para salir del juego, con el que se vuelve a la pantalla de inicio
  5. Un botón para iniciar un nuevo juego, con el que se ingresa a una nueva sala de espera.

Durante el juego si un jugador presiona la barra espaciadora mientras se está jugando el juego se pausará. Cuando alguno de los jugadores gane, para el jugador campeón debe aparecer un mensaje que señale que ganó el juego. Para el resto de los jugadores, debe aparecer un mensaje avisando que perdieron la partida.

Este mensaje puede ser un *label*, una ventana nueva, etcétera. Pero debe permitir que el jugador apriete un botón para salir del juego u otro botón para iniciar una nueva partida.

# 5. *Networking*

Como a (casi) nadie le gusta jugar solo, para estar tarea deberás utilizar tus conocimientos de *networking* para ofrecer el juego «en línea». La implementación de *DCCurve* deberá basarse en una arquitectura de tipo «cliente–servidor», en donde todas las interacciones que ocurran entre usuarios deberán pasar por el servidor. Para lograr lo anterior, deberá implementar un protocolo *eficiente* de comunicación entre el cliente y el servidor usando el modelo TCP/IP. Por otra parte, todas las interacciones que se esperen del cliente deben realizarse a través de las interfaces gráficas.

Como es un juego en línea, en el cual todos los clientes actúan de manera simultánea, es importante que estos reciban la misma información. Por esta razón, todos los usuarios deberán poder ver lo mismo y el servidor deberá actualizar todas las pantallas con cada acción de un cliente.

## 5.1. Archivos

Deberá estar claro cuál es la dirección IP y el puerto que se están ocupando para hacer *host* de las salas de espera y a la cual se conectarán los clientes. Estos deberán ser definidos en una zona visible y escrita en el archivo `README.md`. El cliente y el servidor deberán ser dos archivos distintos que funcionen de manera independiente. En otras palabras, estos no pueden importarse entre ellos. Esto es importante ya que basta con que un computador en la red tenga los archivos de servidor; los demás bastarían con tener sólo los archivos que se enfocan en el cliente. Para esto recomendamos tener dos carpetas: una para el servidor y todos los archivos que este necesite, y otra para el cliente.

# 6. Bonus

Probablemente, quedaste con ganas de seguir programando. No te preocupes: a continuación, te proponemos dos bonus para extender las funcionalidades de esta tarea. Para poder optar a las décimas de este bonus, es requisito **alcanzar al menos un 5,0 en la tarea, sin considerar estas**.

## 6.1. Tres poderes más (5 décimas)

No estando conforme con los poderes actuales, decides agregar **tres** poderes adicionales para hacer el juego aún más entretenido. Para ganar las décimas de este bonus, **todos** los poderes deben estar implementados.

- **Somnolencia Barrios:** Al momento de obtener este poder, el jugador disminuye su rapidez de movimiento a la mitad de su rapidez actual. Luego de cinco segundos, el jugador aumenta su velocidad al doble (*i.e.* volviendo a su rapidez original). Además, las reducciones de velocidad son acumulables.
- **Fernando Atraviessa:** En el momento que cualquier jugador obtenga este poder, todos los jugadores podrán cruzar los límites de la cancha, apareciendo en el lado opuesto de donde atravesaron.
- **Stephanie Tau:** Con este poder, cada vez que el jugador cambie de dirección, lo tendrá que hacer siguiendo un ángulo recto (claro, en **tau**/4 radianes, también conocido como 90°) durante seis segundos.

### 6.2. Modo multijugador (5 décimas)

Este bonus consiste en implementar un modo de dos jugadores dentro del mismo computador. Este modo debería permitir que un usuario pueda jugar con alguien más, además de jugadores a través de *networking*. El segundo jugador deberá moverse con teclas definidas por el mismo jugador —de la misma forma que las tiene que solicitar el primer jugador en el juego normal.

Las funcionalidades se deben mantener iguales: con el cambio de que ahora uno de los jugadores de *lobby* se comporta como *invitado*, no requiere de nombre y registro, por lo que al jugar el nombre que se despliega es «Invitado».

## 7. Notas

Como podrás haber notado, este enunciado no entrega detalles específicos acerca de la interfaz gráfica. Asimismo, los parámetros del juego tampoco están definidos de forma estricta. Esto es así de forma intencional; queda a tu criterio construir un juego *amigable* para el usuario. Mantente dentro de rangos razonables de tal forma que un jugador pueda aprovechar todas las funcionalidades de *DCCurve*.

## 8. Entregable

El propósito de este entregable preliminar es guiar un aspecto de la implementación gráfica de tu tarea. Deberás investigar **al menos** dos técnicas —relacionadas con PyQt5— para realizar el dibujo del rastro de los jugadores dentro del juego. La idea es investigar, comparar y evaluar (*e.g.* mencionar ventajas, desventajas, qué harán para implementarlas) estas técnicas. Luego, tendrás que elegir, de manera justificada, una de ellas. Este breve análisis deberá ser entregado a través de un documento escrito en formato *Markdown*. No es necesario que este documento sea extenso; de hecho, trata de ser lo más conciso posible.

## 9. 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 3.6.
- Si no se encuentra especificado en el enunciado, asume 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 archivo `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. **Tendrás hasta 24 horas después del plazo de entrega** de la tarea para subir el *readme* a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la **guía de descuentos**.
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

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