Tarea 2

Entrega

- Avance de tarea
  - Fecha y hora: miércoles 3 de noviembre de 2021, 20:00
  - Lugar: Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- Tarea
  - Fecha y hora: sábado 13 de noviembre de 2021, 20:00
  - Lugar: Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- `README.md`
  - Fecha y hora: sábado 13 de noviembre de 2021, 22:00
  - Lugar: Repositorio personal de GitHub — Carpeta: `Tareas/T2/`

Objetivos

- Utilizar conceptos de interfaces y PyQt5 para implementar una aplicación gráfica e interactiva.
- Tomar decisiones de diseño y modelación en base a un documento de requisitos.
- Entender y aplicar los conceptos de *back-end* y *front-end*.
- Aplicar conocimientos de *threading* en interfaces.
- Aplicar conocimientos de señales.

Índice

1. *DCCrossy Frog*

Debido a la escasez hídrica que afecta al país, las ranas han visto afectado su hábitat natural. Es por esto que en busca de comida, Froggy persiguió una mosca por mucho tiempo, hasta que sin darse cuenta, había llegado a la ciudad. Estar en este ambiente desconocido por el que pasan autos y ríos, generó que Froggy viera su vida peligrar en incontables ocasiones.

Froggy con mucho miedo por esta situación y con muchas ganas de volver con su familia, te encontró a ti. Luego de contarle que eras un experto programador, se le ocurrió que con tus conocimientos de *threading* e interfaces gráficas le podrías ayudar a volver a su casa sano y salvo. Debido a lo anterior, te pidió que diseñes un programa que le permita practicar el camino de vuelta a su hogar, para que así, pueda memorizarlo y volver a reencontrarse con su familia sin mayores complicaciones.

2. Flujo del programa

DCCrossy Frog es un juego que consiste en ayudar a Froggy a cruzar calles y ríos para que pueda volver a su hogar. Para esto, deberás moverte a través de un mapa que se divide en dos partes. Por la primera pasan autos los cuales deberás esquivar y por la segunda pasan troncos que permiten el movimiento entre ellos y evitan que Froggy caiga al agua. El objetivo final es llegar al final del mapa (parte superior de la pantalla) pudiendo recolectar ciertos objetos en el camino. Al finalizar cada ronda se calculará un puntaje en base a la cantidad de monedas atrapadas, vidas y tiempo restantes.

Al iniciar el programa se mostrará la ventana de inicio, en la que el jugador podrá ingresar su nombre de usuario o abrir una ventana de *rankings* con los mejores puntajes registrados.

Si se elige iniciar el juego, se deberá cerrar la ventana de inicio y abrir la ventana de juego. Esta ventana mostrará el mapa de juego y el personaje. Al comenzar la ronda se moverán los troncos y autos de manera horizontal. También aparecerán, de manera aleatoria, objetos en el mapa, los cuales se podrán recoger para obtener beneficios. Una ronda termina cuando se llega al final, se acaba el tiempo o se pierden todas las vidas.

Una vez que haya finalizado la ronda, se abrirá la ventana de post-nivel en la cual se mostrarán tanto las estadísticas acumuladas como de la ronda. En esta se deberá notificar al jugador si puede seguir jungando o no. En el primer caso, se dará la opción de continuar al siguiente nivel. Por el contrario, en caso de no poder seguir, se deberá informar al jugador su puntaje total, registrándolo en un archivo y se dará la opción de volver a la ventana de inicio.

3. Mecánicas del juego

DCCrossy Frog contiene ciertas mecánicas claves que deben implementarse para un correcto funcionamiento del programa. En esta sección se explica el funcionamiento de las principales mecánicas del juego.

3.1. Personaje

La interacción del usuario con el programa se realizará mediante el personaje del juego llamado Froggy. Este debe atravesar el nivel seleccionado en un tiempo determinado dependiendo de la dificultad del nivel.

Para ello, al iniciar el juego Froggy debe localizarse al principio del nivel sobre una franja de pasto en la parte inferior del mapa, donde no habrán obstáculos. A medida que avance el juego, Froggy podrá moverse libremente en 4 direcciones (arriba, abajo, derecha e izquierda) para esquivar los autos en la Carretera y recolectar los objetos que vayan apareciendo. Además, el personaje podrá realizar un salto discreto hacia adelante para saltar de un tronco a otro en la zona de Río. El movimiento del personaje se explicará con más detalle en la sección Interacción con el usuario.

El personaje tendrá una cantidad `VIDAS_INICIO` de vidas y cada vez que colisione con un auto o se caiga de

un tronco al río, perderá una vida.[1]
En ese momento, Froggy se localizará nuevamente al principio del camino (sección inferior del mapa) en una zona de pasto libre de obstáculos y tendrá que atravesar el nivel entero nuevamente.

3.2. Áreas de juego

Para finalizar el nivel, Froggy tendrá que atravesar dos áreas: la carretera y el río. En cada una de ellas habrá distintos tipos de obstáculos y objetos para recolectar.

Carretera

En esta sección, el personaje tendrá que atravesar la carretera por donde pasan autos de un lado al otro a una velocidad determinada por la Dificultad del nivel.
En la carretera deberán haber 3 hileras de autos, cada una con uno o más autos. Estos últimos deben aparecer y moverse ya sea de derecha a izquierda o de izquierda a derecha (aleatoriamente), pero todos los autos de una misma hilera deben moverse en el mismo sentido. Cuando el extremo final del auto alcance el límite del mapa, el auto deberá desaparecer. Los automóviles deberán aparecer en pantalla cada `TIEMPO_AUTOS` segundos teniendo cuidado de no hacer colisionar dos autos o de no dejar suficiente espacio para que Froggy pueda pasar. Tengan en consideración que este parámetro no será disminuido de ese mínimo, por lo que siempre se respetará el tiempo necesario para que los autos no se acoplen en la misma hilera.
El personaje tendrá que esquivar los autos para poder seguir por el camino. En caso de colisionar con uno de ellos, perderá una vida y deberá volver al inicio del nivel sobre el área de pasto sin ninguna clase de obstáculo.

Río

En esta parte del juego, el personaje tendrá que atravesar el río saltando de un tronco a otro mediante un salto discreto, es decir, un movimiento que salte un grupo de píxeles de una sola vez. Tal como se menciona en la sección de Salto, no es necesario darle animación al salto, pero en caso de querer hacerlo se les ha provisto con *Sprites* de salto para Froggy. La velocidad de los troncos dependerá de la Dificultad del nivel.
Mientras el personaje esté sobre algún tronco, este se debe mover junto a él hasta que salte al próximo tronco. Es decir, si el tronco se mueve en cierta dirección, Froggy seguirá el mismo movimiento junto al tronco. En caso de caer al agua o de colisionar con uno de los bordes de la pantalla, perderás una vida y Froggy tendrá que volver a la franja de pasto al inicio del nivel.
Cabe destacar que se debe verificar de forma explícita que al saltar de un tronco a otro el destino de Froggy sea sobre un tronco y no sobre el río. El personaje podrá moverse de forma libre dentro del tronco, pero sólo podrá salir de este mediante un salto hacia adelante o hacia atrás (no hacia los lados ni en diagonal).
De forma similar a los autos, deberán haber 3 hileras de troncos. Estos deberán moverse de izquierda a derecha o de derecha a izquierda (aleatoriamente), pero la dirección del movimiento de los troncos de cada hilera debe ser de forma intercalada. Por ejemplo, si la primera hilera de troncos se mueven de izquierda a derecha, entonces la segunda hilera se moverá de derecha a izquierda. Luego, la tercera hilera deberá moverse de izquierda a derecha. Al momento en que el extremo del tronco alcance el límite del mapa, deberá volverse invisible al usuario. Los troncos deberán aparecer en pantalla cada `TIEMPO_TRONCOS` segundos, de forma similar a los autos, se debe tener cuidado de no hacer colisionar dos troncos o de dejar demasiados de estos de manera que sea trivial que Froggy pueda pasar. Al igual que en los autos, este parámetro no será disminuido, por lo podrás asegurarte que siempre se podrá respetar el espacio necesario para que los troncos no colisionen al momento de aparecer.

[1] Una forma de comprobar colisiones es utilizando la clase `QRect` de `QtCore` y el método intersects. Este método permite obtener la intersección de dos objetos `QRect`.

3.3. Mapa

El mapa que se mostrará en la Ventana de Juego deberá tener 3 tipos de terreno: pasto, carretera y río. Todo nivel deberá comenzar y finalizar con una zona de pasto que corresponderá al inicio y meta, respectivamente. Además, el mapa debe contar con 3 áreas de juego, con al menos una de carretera y una de río. La tercera queda a su criterio. De esta forma, un mapa se establecerá con dos ríos y una carretera, o dos carreteras y un río.

En cada carretera deberán haber 3 hileras de autos y análogamente, en cada río deberán existir 3 hileras de troncos. Esto quiere decir que es posible observar varios autos o troncos en una hilera, pero un auto o tronco no puede estar entre medio de dos hileras. Además, entre cada área de juego (río o carretera) deberá haber una separación de pasto, a menos que las dos áreas sean del mismo tipo. Es decir, pueden haber dos ríos seguidos o dos carreteras seguidas, pero entre un río y una carretera siempre debe haber pasto. En caso de poner dos carreteras o dos ríos de forma contigua, de manera que exista una carretera o un río con 6 carriles de autos o troncos, se considerarán como dos áreas de juego.

3.4. Objetos especiales

En el juego existirán diferentes objetos especiales, cada uno realizará un efecto especial sobre el juego al colisionar con ellos. Estos objetos deberán aparecer cada `TIEMPO_OBJETO` segundos, es decir, en cada intervalo de esa duración se debe escoger uno de los siguientes objetos de forma aleatoria e insertarlo en el mapa. Se deberá verificar que estos objetos no aparezcan sobre el río ni la meta. En caso de que aparezcan objetos sobre la carretera, los autos deberán atravesarlos sin hacerlos desaparecer.

- Vidas extra: al recoger este objeto se le sumará una vida al personaje.
- Monedas: al recoger este objeto, se le sumará una cantidad `CANTIDAD_MONEDAS` al banco de monedas del jugador. Las monedas deberán ser mostradas junto a las vidas en la ventana de juego y podrán ser utilizadas en caso de realizar el bonus Ventana de Tienda (5 décimas) para comprar vidas y *skins*.
- Calaveras: al recoger este objeto la velocidad de los troncos aumentará en un $5\,\%$ y sólo afectarán a los troncos del nivel actual.
- Relojes: al recoger este objeto se aumenta el tiempo, en segundos, en un valor dado por:

$$tiempo\_adicional = 10 \cdot \left[\frac{tiempo\_restante}{duración\_ronda} \right]$$

  En donde:
  - *tiempo_restante*: es el tiempo que queda de la ronda
  - *duración_ronda*: es la duración total de la ronda, que depende del nivel

3.5. Dificultad

DCCrossy Frog tiene distintos niveles de dificultad que van aumentando a medida que avanza el juego.

Al iniciar el juego, la ronda durará `DURACION_RONDA_INICIAL` segundos, los autos se moverán con una velocidad `VELOCIDAD_AUTOS` y los troncos con una velocidad `VELOCIDAD_TRONCOS`. Cada vez que se pase al siguiente nivel, el tiempo de ronda disminuye por un factor entre 0 y 1 dado por `PONDERADOR_DIFICULTAD` como se especifica en la siguiente fórmula:

$$duracion\_ronda = \texttt{PONDERADOR\_DIFICULTAD} \cdot duracion\_ronda\_anterior$$

Además, las velocidades de los autos y troncos deberán aumentar en cada nivel según las siguientes fórmulas:

$$nueva\_velocidad\_autos = velocidad\_autos\_anterior \cdot \frac{2}{(1 + \texttt{PONDERADOR\_DIFICULTAD})}$$

$$nueva\_velocidad\_troncos = velocidad\_troncos\_anterior \cdot \frac{2}{(1 + \texttt{PONDERADOR\_DIFICULTAD})}$$

3.6. Puntaje y vidas

Una vez finalizado un nivel, el personaje solo podrá pasar al siguiente si es que ganó el nivel anterior. Al completar un nivel se obtendrá un puntaje que se se calculará según la siguiente fórmula:

$$puntaje\_nivel = (vidas \times 100 + tiempo\_restante \times 50) \times nivel$$

En donde:

- *vidas*: son las vidas que le quedan al jugador
- *tiempo_restante*: es el tiempo que queda de la ronda
- *nivel*: es el número de nivel en el que se está jugando[2]

3.7. Fin de nivel

Una vez que el personaje llegue a la meta del camino (parte superior del mapa), pierda todas sus vidas o se le acabe el tiempo, el nivel finalizará y se mostrará la ventana post-nivel con las estadísticas del juego. Solo si el personaje gana el nivel (si llega a la meta con vidas), podrá pasar al siguiente.

3.8. Fin de juego

Si el personaje se queda sin vidas o se le acaba el tiempo antes de llegar al final del camino en la ventana post-nivel se deberá notificar la derrota del jugador. El nombre de usuario y su puntaje acumulado se deberán almacenar automáticamente en el archivo `puntajes.txt`. Luego, el jugador tendrá la opción de volver a la ventana de inicio para iniciar una nueva partida.

4. Interfaz gráfica

4.1. Modelación del programa

Se evaluarán, entre otros, los siguientes aspectos:

- Correcta modularización del programa, lo que incluye una adecuada separación entre *back-end* y *front-end*, y un diseño cohesivo con bajo acoplamiento.
- Correcto uso de señales y *threading*[3] para modelar todas las interacciones en la interfaz.
- Presentación de la información y funcionalidades pedidas (puntajes o avisos, por ejemplo) a través de la interfaz gráfica. Es decir, no se evaluarán *items* que solo puedan ser comprobados mediante la terminal o por código a menos que el enunciado lo explicite.

4.2. Ventanas

Dado que *DCCrossy Frog* se compondrá de diversas etapas, se espera que estas se distribuyan en múltiples ventanas. Por lo tanto, tu programa deberá contener como mínimo las ventanas que serán mencionadas a continuación, junto con los elementos pedidos en cada una. Los ejemplos de ventanas expuestos en esta sección son simplemente para que te hagas una idea de cómo se deberían ver y no esperamos que tu tarea se vea exactamente igual.

[2] El nivel inicial parte en 1

[3] Esto considera el uso de elementos de *threading* de `PyQt5`, como `Qthreads` o `Qtimer`, entre otros.

4.3. Ventana de Inicio

Es la primera ventana que se debe mostrar al jugador al momento de ejecutar el programa. Debe incluir un área donde ingresar el nombre de usuario, un botón para comenzar una nueva partida y un botón para ver el ranking de jugadores.
Si el jugador desea iniciar una partida nueva, se deberá verificar si el nombre de usuario cumple la restricción alfanumérica y tiene un mínimo de `MIN_CARACTERES` y un máximo de `MAX_CARACTERES` (inclusive). En el caso de que no cumpla, se deberá notificar mediante un mensaje de error o *pop-up*. Una vez que se cumpla con lo especificado, la ventana de inicio se cerrará y se dará paso a la ventana de juego.
De otro modo, si el jugador selecciona la opción de ver el *ranking* de puntajes, se deberá mostrar la ventana de *ranking*.

4.4. Ventana de Ranking

Debido a la gran dificultad de *DCCrossy Frog*, es importante dar reconocimiento a sus mejores jugadores. Por esta razón, en esta ventana se deberá mostrar los 5 mejores jugadores que han sido registrados en `puntajes.txt`, desplegando su nombre de usuario y el puntaje obtenido. Los nombres y los puntajes de los jugadores deberán ser ordenados de forma descendente, es decir, el primer puntaje será el más alto y el último puntaje será el más bajo.
Además, se debe poder volver a la ventana de inicio mediante un botón o simplemente cerrando esta ventana.

4.5. Ventana de Juego

En esta ventana se desarrollan las mecánicas del *DCCrossy Frog*. Esta deberá contener el mapa del juego, las estadísticas y las opciones de pausar y salir del juego. Las estadísticas del juego deberán actualizarse a medida que avance el nivel. Deben contener como mínimo:

- Vida actual del jugador
- Nivel actual
- Puntaje obtenido en el nivel
- Tiempo restante para finalizar el nivel
- Cantidad de monedas atrapadas en total

Además, tu mapa debe mostrar la carretera y el río por donde pasará Froggy, junto con espacios de pasto entre ellos. Recuerda que deben existir como mínimo un área de río y una de carretera, siendo la tercer área escogida por ti entre estas dos opciones. Queda a tu criterio cómo implementar el mapa con sus respectivas áreas y las medidas entre cada una de ellas, considerando que se cumpla con lo anteriormente especificado.
Finalmente, si el jugador pierde una vez terminado el tiempo o habiéndose acabado las vidas del jugador (lo que ocurra primero), o completa el nivel, este nivel se terminará y se dará paso a la ventana de post-nivel. En este se guardará el puntaje total obtenido de todos los niveles y deberá registrarse en el archivo `puntajes.txt` siguiendo su formato. En caso de que el jugador haya ganado el nivel, no es necesario guardar el puntaje.

4.6. Ventana de Post - Nivel

Cada vez que se termine un nivel, se deberá mostrar esta ventana. Esta deberá contener un resumen sobre el nivel anteriormente jugado, mostrando:

- Vidas restantes del jugador
- Nivel actual
- Tiempo restante
- Puntaje obtenido en el nivel
- Cantidad de monedas atrapadas en total

- Botones para continuar partida, salir del juego y dirigirse a la ventana de tienda (esta última sólo si se realiza el bonus).

Si el jugador aún tiene vidas para continuar, se deberá notificar con un mensaje en pantalla y se deberán mostrar habilitadas las opciones para continuar jugando el siguiente nivel y para ir a la tienda (en caso de implementar el bonus Ventana de Tienda (5 décimas)). De lo contrario, se deberá notificar al usuario que no puede continuar jugando y se tendrán deshabilitadas los botones anteriormente mencionados. Luego, si se selecciona la opción de continuar partida, se deberá abrir la ventana de juego para comenzar el siguiente nivel, mientras que si escoge ir a la tienda (en caso de implementar este bonus), se deberá abrir la ventana de tienda.

5. Interacción con el usuario

5.1. Movimiento

En tu programa, Froggy debe poseer movimientos animados para avanzar en el mapa. Para lograr esto, debe diferenciarse su aspecto al moverse, considerando al menos tres estados de movimiento para cada dirección como se puede apreciar en los *Sprites* entregados.
El movimiento del personaje se da a través de las teclas `WASD` y no es necesario que implementes movimiento en diagonal. La velocidad de movimiento de Froggy estará indicada por `VELOCIDAD_CAMINAR`, es decir, la cantidad de píxeles que se moverá al presionar una tecla en una dirección específica.

5.2. Salto

Además de permitir que Froggy camine, es importante que Froggy salte para poder atravesar los troncos en el río sin caerse al agua. Para ello, deberás implementar la acción de saltar, la cual se ejecutará mediante el pulso de la barra espaciadora. En caso de tener problemas con el uso de esta tecla, esta puede ser cambiada por otra y será necesario que especifiques la nueva tecla en tu `README.md`.
El salto consiste en mover a Froggy una cantidad de `PIXELES_SALTO` en la dirección en la cual se encuentra. Es decir, si Froggy se encuentra mirando hacia la izquierda y se pulsa la barra espaciadora, se deberá mover el sprite de Froggy `PIXELES_SALTO` pixeles hacia la izquierda.

5.3. Click

Cada vez que quieras interactuar con algún botón o *widget* de tu interfaz, deberás hacer *click* en ellos.

5.4. Cheatcodes

Con la finalidad de ~~facilitar la corrección~~ mejorar la experiencia de juego, es posible presionar un conjunto de teclas en orden, o al mismo tiempo[4], para hacer "trampa" durante cada nivel:

- V + I + D: Esta combinación de teclas aumenta tus vidas en `VIDAS_TRAMPA`.
- N + I + V: Esta combinación de teclas termina el nivel actual y lleva directo a la ventana de post-nivel. Si se usa estando fuera del nivel (por ejemplo en la ventana de post-ronda), no tendrá efecto. De esta manera, el puntaje obtenido y las estadísticas deberán calcularse con el progreso logrado hasta antes de utilizar la combinación.

5.5. Pausa

En la ventana de juego, debe existir un botón de pausa que al ser presionado pause o reanude el juego, el cual también debe ser activable con la tecla P. El juego debe estar visualmente pausado, sin ocultar sus elementos. Durante la pausa no se podrá mover al personaje y se pausarán las demás mecánicas del juego, considerando

[4] Queda a tu criterio, pero deberás especificarlo en tu Readme

el movimiento de troncos, autos y el tiempo. Una vez que se vuelva a presionar la tecla P o se haga *click* sobre el botón de pausa, todos los procesos reanudarán lo que estaban haciendo, sin reiniciarse ni perderse.

6. Archivos

Con la finalidad de que puedas comenzar a jugar *DCCrossy Frog*, deberás hacer uso de los siguientes archivos:

- Los elementos visuales, que están contenidos en la carpeta `sprites` y que se encuentran descritos en *Sprites*. Puedes utilizar tanto los sprites entregados junto al enunciado, o crear tus propios sprites para la tarea.
- Los archivos de audio para el bonus, que están contenidos en la carpeta `canciones`, descritos en Canciones.

Adicionalmente, será tu deber:

- Generar y actualizar un archivo `puntajes.txt`.
- Crear y utilizar un archivo `parametros.py`.

6.1. *Sprites*

Esta carpeta contiene múltiples subcarpetas con imágenes en formato `.png` que te serán útiles:

- `Personajes:` Contiene diferentes carpetas con los *sprites* de movimiento para cada personaje. El nombre de cada imagen indicará el movimiento correspondiente con el formato `{movimiento}_{valor}`, donde movimiento podrá tomar los valores `right`, `left`, `up`, `down` y `jump`, y valor podrá ser un número entre 1 y 3. Además, se encuentra el sprite `still`, que corresponde al personaje quieto, en caso de que se requiera utilizarlo.
- `Mapa:` Contiene carpetas con las imágenes que te ayudarán a implementar cada mapa del juego, incluyendo los *sprites* de los autos que intentarán atropellarte y los troncos del río. Además, contarás con los *sprites* de la carretera, pasto y el río para que puedas construir tu mapa de juego. Por último, se incluyen elementos gráficos como faroles y árboles en caso de que quieras adornar tu mapa de juego, pero no es necesario utilizarlos, ni tienen un puntaje asociado.
- `Objetos:` Contiene imágenes para los objetos que aparecerán en el mapa durante la travesía de Froggy.

6.2. Canciones

En caso de que desees realizar el bonus Música (3 décimas), podrás animar el ambiente de de *DCCrossy Frog*, permitiendo que Froggy salte al ritmo de la música. En la carpeta `canciones` encontrarás la canción que podrás reproducir dentro de tu programa.

Puedes probar tu tarea usando tus propias canciones, sin embargo, debes ignorar todos los archivos con formato `WAV` al momento de subir tu tarea a tu repositorio remoto.

6.3. `puntajes.txt`

Este archivo se encargará de mantener un registro de todos aquellos jugadores que han tenido el honor de jugar una partida de *DCCrossy Frog*. Por cada partida terminada se debe almacenar una fila con el formato usuario, puntaje, los cuales podrás almacenar en el orden que estimes conveniente. Es importante mencionar que este archivo debe contener un registro de todos los puntajes que fueron almacenados. Sin embargo, en la Ventana de Ranking deberás mostrár solo los 5 primeros en orden descendente.

Tu tarea también deberá encargarse de crear el archivo en caso de no existir al momento de ejecución. Un ejemplo de los contenidos del archivo es el siguiente:

```
mmunoz12,1700
tamyhan,1600
ivanvlam,7000
sebaguerrap,1000
cegalleta,1300
matiasmasjuan,0
```

6.4. `parametros.py`

A lo largo del enunciado se han ido presentando distintos números y palabras en `ESTE_FORMATO`. Estos son parámetros del programa y son valores que permanecerán constantes a lo largo de toda la ejecución de tu código.

En este archivo se deben encontrar todos los parámetros mencionados en el enunciado, además de todos los *paths* y cualquier otro valor constante que vayas a utilizar en tu código. Cada línea de este archivo debe almacenar una constante junto a su respectivo valor. Asegúrate que los nombres de las constantes sean descriptivos y fáciles de reconocer.

Este archivo debe ser importado como un módulo y así utilizar sus valores almacenados. En caso de que no se especifique el valor de un parámetro en el enunciado, deberás asignarlo a tu criterio, procurando que no dificulte la interacción con el programa.

Es posible que los ayudantes modifiquen estos parámetros durante la corrección, pero los que hagan referencia a dimensiones de ventanas, dimensiones de elementos de la interfaz o rutas a *sprites* no serán modificados para no complicar la visualización de estos. Sin embargo, deberás agregarlos como parámetros al archivo de igual modo.

7. *Bonus*

En esta tarea habrá una serie de *bonus* que podrás obtener. Cabe recalcar que necesitas cumplir los siguientes requerimientos para poder obtener *bonus*:

1. La nota en tu tarea (sin bonus) debe ser igual o superior a 4.0.[5]
2. El bonus debe estar implementado en su totalidad, es decir, no se dará puntaje intermedio.

Finalmente, la cantidad máxima de décimas de *bonus* que se podrá obtener serán 10 décimas. Deberás indicar en tu `README` si implementaste alguno de los bonus, y cuáles fueron implementados.

7.1. Ventana de Tienda (5 décimas)

En esta ventana se deberán mostrar los objetos disponibles para comprar por lo jugadores. A cambio de las monedas reunidas en cada nivel, el jugador podrá obtener vidas y diferentes *skins* para su rana que se encuentran a la venta. Estas *skins* deberán cambiar el *sprite* de Froggy, las cuales podrán ser encontradas en la carpeta `Personajes` dentro de `sprites`. Deberás mostrar en tienda las opciones de comprar *skin* naranja, roja o verde.

La cantidad de monedas necesarias para comprar cada uno de los objetos deberá ser almacenada en el archivo `parametros.py`, por lo que deberás crear tus propios parámetros para ello. Si el jugador no posee la cantidad de monedas necesarias para comprar el objeto deseado, se deberá notificar mediante un mensaje o *pop-up*. Por otro lado, si se tienen suficientes monedas, entonces se deberá descontar el monto asociado a la cantidad de monedas que posee el usuario y se deberá añadir la vida comprada o cambiar el *skin* de Froggy, según sea el caso.

[5] Esta nota es sin considerar posibles descuentos.

Cada vez que se desee comprar o cambiar de *skin*, se deberá pagar el precio correspondiente a esta. Es decir, cada vez que se cambie de *skin*, se deberá comprar nuevamente y descontar su precio asociado, sin importar de si se había comprado anteriormente.
Finalmente, se deberán implementar los botones para abrir la tienda en la ventana de post-nivel. Se deberá poder volver a la ventana de post-nivel mediante un botón de volver o simplemente cerrando esta ventana.

7.2. Música (3 décimas)

Para animar un poco tu programa, decides incluir *música* en este. Para esto, deberás utilizar como música el archivo `WAV` entregado junto al enunciado.
La música debe ejecutarse durante todo momento en el programa, reiniciándose cada vez que se cambie de ventana. Además, cuando se pause el programa, la música debe detenerse y reanudarse una vez el programa se reanude.
La reproducción de esta canción se deberá hacer mediante la librería `PyQt5`[6], por lo que el uso de cualquier otra librería de Python para reproducirlas será considerado como inválido y no obtendrá el puntaje respectivo.

7.3. Checkpoint (2 décimas)

Después de morir muchas veces te das cuenta de que el juego sería mucho más entretenido si hubiese un *checkpoint* al cual volver cuando se pierde una vida.
Para lograr este bonus se debe incluir al menos un *checkpoint* en alguna sección de pasto (sin considerar la inicial ni la final) de tal forma que al perder una vida Froggy vuelva a esa sección de pasto. Por ejemplo, si al comienzo del mapa Froggy cruza la carretera hasta llegar a un pasto, este será el nuevo punto de inicio en caso de que Froggy pierda una vida más adelante.

8. Avance de tarea

En esta tarea, el avance corresponderá a manejar apropiadamente dos tipos diferentes de colisiones y realizar el movimiento de Froggy.
Para ello, debes crear una ventana en donde se encuentre Froggy, un objeto especial y un auto. El personaje debe poder moverse según lo indicado en el enunciado, es decir, mediante las flechas `WASD` y utilizando diferentes *sprites* para simular un movimiento fluido. No es necesario implementar el movimiento de salto.
En caso de que el personaje colisione con el objeto especial, este último debe ser recolectado y desaparecer de la ventana. Por otro lado, en caso de que colisione con el auto, el personaje debe trasladarse al punto inicial y el auto debe permanecer en la ventana. No es necesario implementar el movimiento del auto.
A partir de los avances entregados, se les brindará un *feedback* general de lo que implementaron en sus programas y además, les permitirá optar por hasta 2 décimas adicionales en la nota final de su tarea.

9. `.gitignore`

Para esta tarea deberás utilizar un `.gitignore` para ignorar los archivos indicados, este deberá estar dentro de tu carpeta `Tareas/T2/`. Puedes encontrar un ejemplo de `.gitignore` en el siguiente *link*.
Los archivos a ignorar para esta tarea son:

- Enunciado
- Carpeta de *sprites* entregada junto al enunciado.
- Carpeta de *música* entregada junto al enunciado.

[6] Las clases `QSound` o `QMediaPlayer` podrían serte útiles.

Recuerda no ignorar tu archivo de parámetros, o tu tarea no podrá ser revisada.
Se espera que no se suban archivos autogenerados por las interfaces de desarrollo o los entornos virtuales de Python, como por ejemplo: la carpeta `__pycache__`.
Para este punto es importante que hagan un correcto uso del archivo `.gitignore`, es decir, los archivos no deben subirse al repositorio debido al archivo `.gitignore` y no debido a otros medios.

10. Entregas atrasadas

Posterior a la fecha de entrega de la tarea se abrirá un formulario de Google Form, en caso de que desees que se corrija un *commit* posterior al recolectado, deberás señalar el nuevo *commit* en el *form*.
El plazo para rellenar el *form* será de 24 horas, en caso de que no lo contestes en dicho plazo, se procederá a corregir el *commit* recolectado.

11. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del programa será el pilar de la corrección, es decir, sólo se corrigen tareas que se puedan ejecutar. Por lo tanto, se recomienda hacer periódicamente pruebas de ejecución de su tarea y *push* en sus repositorios.
Cuando se publique la distribución de puntajes, se señalará con color:

- Amarillo: cada ítem que será evaluado a nivel de código, todo aquel que no esté pintado de amarillo significa que será evaluado si y sólo si se puede probar con la ejecución de su tarea.
- Azul: cada ítem en el que se evaluará el correcto uso de señales. Si el ítem está implementado, pero no utiliza señales, no se evaluará con el puntaje completo.

En tu archivo `README.md` deberás señalar el archivo y la línea donde se encuentran definidas las funciones o clases relacionados a esos ítems.
Finalmente, 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 jefe de Bienestar al siguiente correo: bienestar.iic2233@ing.puc.cl.

12. Restricciones y alcances

- Esta tarea es estrictamente individual, y está regida por el Código de honor de Ingeniería.
- Tu programa debe ser desarrollado en Python 3.8.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- 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 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 2 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 $(1{,}0)$.