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

IIC2233 Programación Avanzada (2021-1)

Tarea 2

Entrega

- **Avance de tarea**
  - **Fecha y hora:** miércoles 26 de mayo de 2021, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- **Tarea**
  - **Fecha y hora:** sábado 5 de junio de 2021, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- `README.md`
  - **Fecha y hora:** lunes 7 de junio de 2021, 20:00
  - **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. *DCCimpsons*

A causa del Progravirus, los habitantes de Springfield, al igual que tú, se encuentran encerrados haciendo cuarentena en sus casas. Para poder facilitar el tránsito de personas, la DCComisaría Virtual del Jefe Górgory ha decidido habilitar la opción para solicitar Permisos Temporales de manera *online*. Esto permitirá que todos los ciudadanos puedan salir durante un corto periodo de tiempo a abastecerse de sus productos esenciales.

Homero, un conocido de tu infancia, se dio cuenta que el tiempo de cada permiso es muy ajustado como para poder comprar la cantidad de ~~donas~~ insumos básicos que necesita para la semana. Es por esto que te pidió ayuda a ti para que, utilizando tus conocimientos de *threading* e interfaces gráficas, le ayudes a crear un juego que le permita practicar sus salidas y así poder realizarlas lo más rápido posible.

Figura 1: Logo de *DCCimpsons*

2. Flujo del programa

DCCimpsons es un juego que consiste en recolectar la mayor cantidad de objetos en un tiempo determinado. Para esto, deberás moverte a través de un mapa esquivando obstáculos que se encuentran repartidos por la ciudad. Al finalizar cada ronda, se te asignará un puntaje según la cantidad de objetos atrapados en esta, siendo el objetivo del juego obtener la mayor puntuación posible.

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 las mejores puntuaciones registradas.

En caso de iniciar el juego, se deberá cerrar la ventana de inicio y abrir una **ventana de preparación**. Esta ventana mostrará un mapa de la ciudad, donde el jugador tendrá la opción de seleccionar una dificultad para el juego y un **personaje** con el que se podrá mover a través de esta, lo que le permitirá ingresar al edificio correspondiente a este para iniciar una ronda.

Una vez que el jugador haya ingresado a un edificio, se abrirá una **ventana de juego**, en donde se mostrará el tablero de juego y el personaje. Al comenzar la ronda, aparecerán y desaparecerán constantemente, y ubicándose de manera aleatoria, **objetos** que deberá recolectar el jugador, los cuales tendrán una imagen distinta según el personaje que haya sido seleccionado. De la misma manera, se distribuirán aleatoriamente **obstáculos** y **objetos peligrosos** a lo largo del mapa, que tendrán distintos efectos sobre el personaje. En cada ronda también se encontrará el **enemigo**, cuyo objetivo será perseguir al jugador e intentar alcanzarlo para hacerle perder la partida. Una ronda acaba cuando se acabe el tiempo o la vida del jugador.

Una vez finalizada la ronda, aparecerá una **ventana post ronda**, en la cual se mostrará la puntuación alcanzada por el jugador. En caso de poder continuar jugando, se dará la opción de volver a jugar, o regresar a la **ventana de preparación** para cambiar de dificultad y/o personaje. En caso de haber perdido, se mostrará el puntaje alcanzado y se dará la opción de salir del juego.

3. Mecánicas del juego

DCCimpsons 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. Personajes

La principal interacción con *DCCimpsons* se realiza mediante un **personaje** controlado por el usuario, el cual podrá interactuar con distintos ítems en el transcurso de este. El personaje se puede mover libremente en el mapa de la **ventana de preparación** y el de la **ventana de juego**, respetando los márgenes de estas y los Obstáculos.

DCCimpsons posee dos personajes que pueden ser elegidos para jugar en la **ventana preparación**, los cuales son **Homero** y **Lisa**. Cada personaje tiene diferentes velocidades de movimiento, habilidades y Objetos característicos que deben atrapar, y poseen un atributo de **vida**, el cual se comporta como un *float* entre **0** y **1**. Cada personaje comienza el juego con vida máxima, es decir, con un 100 % de ella.

El personaje deberá comenzar en una posición aleatoria dentro del **tablero de juego**, pero debes considerar que su posición inicial no debe coincidir con la de ningún obstáculo. El movimiento de cada protagonista se explicará en mayor detalle en la sección Interacción con el usuario.

Homero

Se mueve con una velocidad de `VELOCIDAD_HOMERO` y su objeto característico son las **donas**.

Su habilidad especial corresponde a que cada vez que atrape `10` donas **Normales**, su vida aumente en una cantidad dada por `PONDERADOR_VIDA_HOMERO`. En caso de que tenga la vida al máximo, esta se debe mantener en 100 %. Además, en caso de colisionar con un objeto **Bueno**, este no se considerará para la cuenta de donas, pero si se colisiona con uno **Peligroso**, esta cuenta se reinicia.

Lisa

Se mueve a una velocidad de `VELOCIDAD_LISA` y su objeto característico son los **saxofones**.

Su habilidad especial permite que los saxofones **Normales** duren una cantidad de `PONDERADOR_TIEMPO_LISA` segundos más en pantalla.

(a) *Sprites* de Homero

(b) *Sprites* de Lisa

Figura 2: Ejemplo *sprites* personajes

3.2. Lugares de juego

Cada personaje posee un destino específico en la **ventana de preparación**, el cual corresponde a un *sprite* de un edificio en la ventana. Al colisionar con este, el programa debe avanzar a la **ventana de juego**.

Cada protagonista **únicamente** puede ir a su edificio designado. Es decir, si colisiona con el otro edificio, el programa no le permitirá comenzar y se debe mostrar en pantalla una notificación informando al respecto[1].

Planta Nuclear

Homero debe ir a ~~dormir~~ trabajar a la Planta Nuclear para poder avanzar a la **ventana de juego**.

Escuela Primaria de Springfield

Al utilizar a Lisa se desbloquea ~~Zoom~~ la Escuela Primaria de Springfield, donde puede avanzar a la **ventana de juego**.

(a) *Sprite* de Planta Nuclear

(b) *Sprite* de Escuela Primaria

Figura 3: Ejemplo *sprites* lugares

3.3. Objetos

En *DCCimpsons*, existen distintos objetos que el personaje podrá recoger durante una ronda. Estos pueden ser **objetos normales**, **objetos buenos** u **objetos peligrosos**, donde cada tipo de objeto tendrá diferentes efectos sobre el juego.

Durante el transcurso de una ronda, cada `APARICION_{DIFICULTAD}`[2] segundos, deberá aparecer **un objeto** en un lugar aleatorio dentro del **tablero de juego**, manteniéndose en la interfaz por una cantidad `TIEMPO_OBJETO_{DIFICULTAD}` de segundos antes de desaparecer. Para saber qué tipo de objeto será el que debe aparecer en cada intervalo de tiempo, se deberá realizar un cálculo usando las probabilidades de aparición de cada objeto [3].

3.3.1. Objetos normales

Los objetos normales corresponden a objetos que deben ser atrapados por el personaje para obtener puntaje durante una ronda. Atrapar un objeto normal, se obtiene una cantidad `PUNTOS_OBJETO_NORMAL` de puntos y tienen una probabilidad `PROB_NORMAL` de aparecer.

Dependiendo del personaje seleccionado, el *sprite* del objeto normal cambiará. En el caso de Homero, serán **donas** y en el caso de Lisa, serán **saxofones**.

[1] Una manera de realizar eso podría ser mediante un `QMessage` o un `QLabel`.

[2] Cuando se muestren parámetros de la forma `PARAM_{DIFICULTAD}`, debes reemplazar la palabra "dificultad"por la dificultad correspondiente (se especifican en la sección Dificultad).

[3] Debido a esto, las probabilidades de aparición de cada tipo de objeto deben sumar 1.

(a) *Sprite* de objeto normal de Homero

(b) *Sprite* de objeto normal de Lisa

Figura 4: Ejemplo *sprites* objetos normales

3.3.2. Objetos buenos

Además de los objetos normales, existirán objetos catalogados como **buenos**, los cuales causarán un efecto positivo sobre el jugador. Los objetos buenos tienen una probabilidad `PROB_BUENO` de aparecer, y en caso de aparecer un objeto bueno, el tipo de este será seleccionado de forma aleatoria. Los tipos de objetos buenos son:

- **Puntos X2**: Al recoger un objeto de este tipo, este entregará el **doble de puntos** que un objeto normal. Su *sprite* será el mismo objeto Normal de cada personaje, pero pintado en **gris**.
- **Aumento de vida**: Al recoger este objeto, se aumenta la vida del personaje en un valor equivalente a `PONDERADOR_CORAZON`. Su *sprite* será un corazón.

Figura 5: *Sprite* de objeto bueno: Aumento de vida

3.3.3. Peligrosos

En el juego existe un objeto peligroso que causará efectos negativos sobre el personaje. Este corresponde al **Veneno**, el cual al ser recogido disminuirá la vida del personaje en un valor equivalente a `PONDERADOR_VENENO`. Su probabilidad de aparecer es de `PROB_VENENO`.

Figura 6: *Sprite* de objeto peligroso: Veneno

3.4. Obstáculos

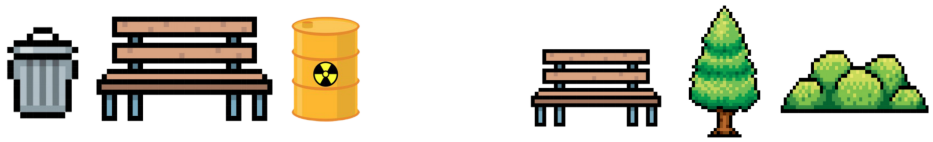
Los **obstáculos** son objetos que aparecen en el **tablero de juego** y bloquean el paso del personaje. Es decir, son lugares por donde el personaje no puede caminar. En particular, al colisionar **cualquier parte del personaje** con alguno de estos, se debe mostrar como si el personaje siguiera caminando, pero sin cambiar su posición.

Al iniciar una ronda, se deberá posicionar uno de cada tipo de obstáculo de forma aleatoria en el **tablero de juego** y estos deberán permanecer en su posición hasta que esta termine. Es importante considerar que un objeto y un obstáculo no podrán aparecer en el mismo lugar. Finalmente, no pueden haber obstáculos posicionados de manera contigua, es decir, deberá haber por lo menos un espacio de separación entre ellos que permita al personaje transitar.

Los *sprites* de los objetos son propios de cada edificio. En otras palabras, dependiendo del personaje elegido, se deben mostrar diferentes *sprites* en el tablero correspondiente. A continuación se muestran los obstáculos propios de cada edificio:

- **Planta nuclear**: Basurero, banca y barril de desechos tóxicos.

- **Primaria**: Banca, árbol y arbusto.



(a) Obstáculos de la Planta nuclear (b) Obstáculos de la Primaria

Figura 7: Ejemplo *sprites* de los obstáculos

3.5. Personaje enemigo

Además del personaje principal, objetos y obstáculos, existirá un personaje enemigo... ¡El **Jefe Górgory**!

Este deberá aparecer en el mismo lugar y replicar el mismo trayecto del personaje, pero con un *delay*/retraso, por lo que aparecerá y comenzará a moverse `TIEMPO_DELAY_{DIFICULTAD}` segundos después que el personaje. Por ejemplo, si el personaje se mueve a la posición (x, y), luego a (x_1, y_1) y después a (x_2, y_2), el Jefe Górgory se seguirá el mismo trayecto $(x, y) \rightarrow (x_1, y_1) \rightarrow (x_2, y_2)$, pero `TIEMPO_DELAY_{DIFICULTAD}` segundos después de que lo haga el personaje.

Si el enemigo alcanza al personaje y colisiona con este, el personaje perderá toda su vida y te llevará a la DCComisaría Virtual, significando el fin del juego.

Figura 8: *Sprite* del Jefe Gorgory

3.6. Dificultad

DCCimpsons posee dos niveles de dificultad, la cual podrá ser seleccionada por el jugador en la **Ventana de preparación**. Las dificultades son **Intro** y **Avanzada**.

- **Intro**: En este nivel, las rondas durarán una cantidad `DURACION_INTRO` segundos y deben aparecer nuevos objetos cada `APARICION_INTRO` segundos.
- **Avanzada**: En este nivel, las partidas duran `DURACION_AVANZADA` segundos y deben aparecer nuevos objetos cada `APARICION_AVANZADA` segundos.

3.7. Puntaje y vida

Una vez finalizada una ronda, el personaje solo podrá pasar a la siguiente si sigue con vida, es decir, su vida es mayor a 0. Al completar una ronda se obtendrá un **puntaje**, el cual se acumulará y te permitirá subir en el *ranking* del juego.

El puntaje de cada ronda se calculará utilizando la siguiente fórmula:

$$puntaje\_ronda = (puntaje\_objetos\_normales + puntaje\_objetos\_X2) \times vida\_personaje$$

En donde:

- *puntaje_objetos_normales*: es el puntaje total obtenido por recolectar objetos normales en la ronda.
- *puntaje_objetos_X2*: es el puntaje total obtenido por recolectar objetos X2 en la ronda.
- *vida_personaje*: valor entre 0 y 1, que representa la vida del personaje al terminar la ronda.

3.8. Fin de ronda

La ronda termina al transcurrir el tiempo de esta, o cuando el personaje se queda sin vida. En este momento se mostrará la **ventana post-ronda** con las estadísticas del juego.

3.9. Fin del juego

Si antes de terminar una ronda el personaje se queda sin vida, en la **ventana post-ronda** se deberá notificar de la derrota del jugador. El nombre de usuario y su puntaje acumulado se deberán almacenar automáticamente en el archivo de `ranking.txt` y el jugador tendrá la opción de volver a la **ventana de inicio** para comenzar 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***[4] 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

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, por lo que no esperamos que tu tarea sea exactamente igual.

4.2.1. Ventana de inicio

Es la primera que se debe mostrar al jugador al ejecutar el programa. Debe incluir como mínimo:

- Área para ingresar el nombre del jugador, con restricción alfanumérica.
- Opción para iniciar una partida nueva.
- Opción para ver el *ranking* de puntajes.

Si el jugador desea **iniciar una partida nueva**, se deberá verificar si el nombre de usuario cumple la restricción alfanumérica. En el caso de que no cumpla, se deberá informar mediante un mensaje el error.

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

Finalmente, una vez que se cumpla con lo especificado, la ventana se cerrará y se dará paso a la **ventana de preparación**. De otro modo, si el jugador selecciona la opción de **ver el *ranking* de puntajes**, se deberá mostrar la **ventana de *ranking***.

Figura 9: Ejemplo de ventana de inicio de *DCCimpsons*

4.2.2. Ventana de *ranking*

En esta ventana se deberán mostrar los 5 mejores puntajes que se hayan registrado en el juego. Se deberá incluir el nombre de los usuarios, junto a sus respectivos puntajes. También se deberá poder volver a la **ventana de inicio** mediante un botón o simplemente cerrando esta ventana.

Figura 10: Ejemplo de ventana de *ranking* de *DCCimpsons*

4.2.3. Ventana de preparación

Esta ventana se ejecuta una vez que el jugador ingrese correctamente con su usuario en una partida nueva.

En esta ventana el jugador podrá seleccionar la **dificultad** del juego y su **personaje**. También se deberá mostrar el **mapa inicial**, en donde aparecerá el personaje una vez seleccionado. Este podrá moverse libremente para llegar a su respectivo edificio y así dar inicio a la ronda.

Además, se deberá incluir en la interfaz una sección que muestre la **vida actual del personaje**, **cantidad de ítems buenos atrapados**, **cantidad de ítems malos atrapados** y las rondas que se hayan completado hasta el momento.

Una vez que el personaje colisione con su edificio correspondiente, se deberá cerrar la ventana y dar paso a la **ventana de juego** para comenzar la ronda.

Figura 11: Ejemplo de ventana de preparación

4.2.4. Ventana de juego

En esta ventana se desarrollan las rondas del juego. En esta ventana se mostrará el tablero correspondiente a la ubicación con la cual se colisionó en la ventana anterior, en donde el usuario jugará su ronda. Además, se deberá incluir una sección con las estadísticas actuales del juego, las cuales incluyen como mínimo:

- Vida actual del jugador.
- Número de ronda actual.
- Tiempo restante para finalizar ronda.
- Cantidad de ítems buenos atrapados.
- Cantidad de ítems malos atrapados.
- Puntaje actual.

Todas las estadísticas mencionadas anteriormente deberán actualizarse a medida que avance el juego.

Finalmente, esta ventana debe tener la opción de **pausar** y **salir** del juego. Una vez terminado el tiempo o las vidas del jugador, la ronda se terminará y se dará paso a la **ventana de post-ronda**.

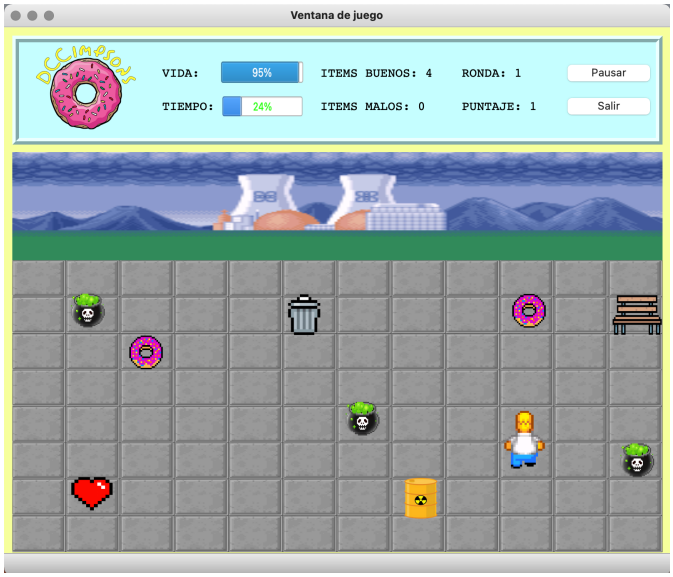


Figura 12: Ejemplo de ventana de juego para el personaje de Homero

4.2.5. Ventana de post-ronda

Al finalizar una ronda, se deberá mostrar esta ventana, que contendrá un resumen sobre la ronda. Esta debe incluir como como mínimo:

- Puntaje acumulado total del jugador.
- Porcentaje de vida restante del jugador.
- Cantidad de ítems buenos atrapados.
- Cantidad de ítems peligrosos atrapados.
- Mensaje indicando si el jugador perdió o puede continuar jugando.
- Opción de salir del juego.
- Si puede seguir, una opción para “Salir” y una opción para “Continuar jugando”.

En caso de seleccionar la opción **continuar jugando**, se deberá abrir nuevamente la **ventana de preparación**, y continuar con el flujo del programa.

Figura 13: Ejemplo de ventana de post-ronda de *DCCimpsons*

5. Interacción con el usuario

5.1. Movimiento

Los personajes deben poseer movimientos animados para avanzar. 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.

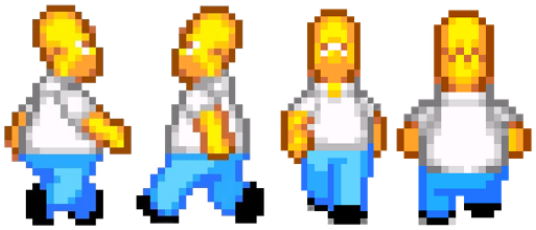


Figura 14: Ejemplos de *sprites* de movimiento para Homero

Puedes usar *sprites* buscadas en internet, usar las entregadas junto con la tarea, o crear tus propias *sprites*. Para más información respecto a las *sprites* entregadas, puedes revisar la sección Archivos.

El movimiento del personaje se da a través de las teclas `WASD` y no es necesario que implementes movimiento en diagonal.

Cada personaje cuenta con una velocidad de movimiento que indicará la cantidad de píxeles que se moverá al presionar una tecla.

5.2. *Click*

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

5.3. *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[5], para hacer "trampa" durante la ronda:

- **V + I + D**: Esta combinación de teclas aumenta tu vida en `VIDA_TRAMPA`.
- **N + I + V**: Esta combinación de teclas termina la ronda actual, lo que hace que los objetos dejen de aparecer. Si se usa estando fuera de una ronda, no tendrá efecto. Por otro lado, el puntaje obtenido y las estadísticas deberán calcularse con el progreso logrado hasta antes de utilizar la combinación.

5.4. 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 no aparecerán ni desaparecerán los objetos. 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.

[5] Queda a tu criterio, pero deberás especificarlo en tu README.md

6. Archivos

Deberás hacer uso de los siguientes archivos:

- Los elementos visuales, que están contenidos en la carpeta `sprites` y que se encuentran descritos en *Sprites*.
- Los archivos de audio para el **bonus**, que están contenidos en la carpeta `canciones`, descritos en Canciones.

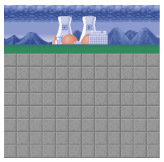
Adicionalmente, será tu deber:

- Generar y actualizar un archivo `ranking.txt`.
- Crear y utilizar un `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` y `down`, y `valor` podrá ser un número entre 1 y 3.
- **`Mapa:`** Contiene carpetas con las imágenes que te ayudarán a implementar cada mapa del juego, incluyendo el mapa de la Ventana de preparación.



(a) *Sprite* de Mapa de Planta Nuclear

(b) *Sprite* de Obstáculo de Planta Nuclear

Figura 15: Ejemplo de *sprites* de carpeta mapa

- **`Objetos:`** Contiene imágenes para los objetos que aparecerán en el mapa durante las rondas.

6.2. Canciones

En caso de que desees realizar el **bonus** de animar el ambiente de *DCCimpsons* con música, esta carpeta contendrá la canción a usar en formato `.wav` que podrás reproducir en 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. `ranking.txt`

Este archivo se encargará de mantener un registro de todos aquellos jugadores que han tenido el honor de jugar una partida de *DCCimpsons*. Por cada partida terminada se debe almacenar una fila con el formato **usuario, puntaje**, los cuales podrás almacenar en el orden que estimes conveniente. 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:

```
lily4167,1000
Dnpoblete,500
DCCollao,1400
Gatochico,700
igbasly,600
```

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 usar 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**[6].
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 8 décimas. Deberás indicar en tu `README` si implementaste alguno de los bonus, y cuáles fueron implementados.

7.1. *Drag and Drop* (5 décimas)

Al seleccionar un personaje en la **Ventana de preparación**, en vez de seleccionarlos mediante algún botón u otro *widget*, deberás usar *drag and drop* para arrastrar al personaje a su posición deseada en el mapa de juego. Al soltar dicho objeto, este debe quedar en una **posición válida**, es decir, no puede quedar por encima de un edificio, o fuera del mapa. En caso de que se intente lo anterior, no se debe permitir colocar al personaje. Una vez se suelte el personaje en una posición válida, si había previamente un *sprite* del personaje en el mapa este deberá eliminarse de la ventana, de forma que solo se mantenga el *sprite* del personaje recién arrastrado.

[6] Esta nota es sin considerar posibles descuentos.

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`[7], 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. Otros personajes (3 décimas)

Tras jugar innumerables partidas de *DCCimpsons*, te has aburrido de usar siempre los mismos dos personajes. Es por esto, que decides programar nuevos personajes con los que jugar tus rondas, junto a sus propios lugares de juego. A continuación se describen los dos personajes y lugares que debes incluir en el programa:

Moe

Moe se mueve con una velocidad de `VELOCIDAD_MOE` y su objeto característico es una cerveza. Su habilidad especial corresponde a disminuir el tiempo que tarda un objeto nuevo en aparecer en el mapa a la mitad.

Al obtener a Moe, para jugar una ronda debes entrar a su Bar en la **Ventana de preparación**.

Krusty

Krusty el payaso posee una velocidad `VELOCIDAD_KRUSTY` de movimiento y su objeto característico es el logo de su restaurante. Su habilidad especial consiste en aumentar el tiempo de *delay* de aparición del enemigo al doble.

Al elegir a Krusty, se debe ingresar al restaurante Krusty Burger en la **Ventana de preparación** para iniciar la ronda de juego.

(a) *Sprite* de Moe

(b) *Sprite* de Krusty

8. Avance de tarea

En esta tarea, el avance corresponderá a **manejar apropiadamente dos tipos diferentes de colisiones con un personaje**.

Para ello, debes crear una ventana en donde se encuentre **el personaje**, un **objeto normal** y un **obstáculo**. 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.

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

En caso de que el personaje colisione con el objeto normal, este último debe ser **recolectado y desaparecer de la ventana**. En cambio, en caso de que colisione con el obstáculo, el personaje **no debe poder seguir avanzando** y el obstáculo **debe permanecer en la ventana**.

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.

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 Forms, 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 a su **correo**.

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 3.7.
- 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 48 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).