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

IIC2233 Programación Avanzada (2023-1)

Tarea 2

Entrega

- **Tarea** y `README.md`
  - **Fecha y hora:** martes 23 de mayo de 2023, 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. *DCCazafantasmas*

Después de tu gloriosa excavación en el torneo **DCCavaCava** y haber ayudado a **Sir Hernan4444** a vencer al malvado **Dr. Pinto**, te dispones a celebrar y disfrutar de esta importante victoria. Sin embargo, esta fiesta no durará por mucho tiempo, ya que de pronto recibes la visita del despavorido **Rey Joaking**, contándote que el malvado **Dr. Pinto** ha regresado. Ante la furia de haber cavado un ~~hoyo~~ pozo menos profundo que **Sir Hernan4444**, nuestro villano ha decidido atemorizar a los habitantes del bosque y las poblaciones aledañas con múltiples criaturas paranormales bajo su poder, secuestrando en el acto al mejor amigo del **Rey Joaking**, el gran mago **Aaossa**.

Luego de esta terrible noticia, decides emprender un viaje junto al gran **Rey Joaking** para rescatar a su amigo. Durante el trayecto, escuchas rumores acerca de que el malvado **Dr. Pinto** ha secuestrado y alojado al mago **Aaossa** en su abandonada mansión embrujada, custodiada por múltiples fantasmas y trampas ocultas para asustar a sus invasores.

Es por esta razón que el gran **Rey Joaking** te pide ayuda a ti, como experto programador, para crear una interfaz donde pueda entrenar hasta el cansancio, utilizando como avatar al mismísimo ~~Mario verde~~ **Luigi** para prepararse ante las múltiples trampas ubicadas en la mansión y lograr salvar a su amigo, el mago **Aaossa**.

Figura 1: Logo de *DCCazafantasmas*

2. Flujo del programa

DCCazafantasmas es un juego que consiste en construir distintos niveles con la finalidad de prepararse ante cualquier escenario posible dentro de la mansión del malvado **Dr. Pinto** y poder rescatar exitosamente al mago **Aaossa**. Para ello, tienes la opción de construir diferentes mapas a partir de 6 tipos de Entidades, las cuales serán posicionadas dentro de ellos para elaborar tus propios niveles. Esto implica que el programa será del tipo **constructor**, vale decir, primero se **personaliza** el mapa y luego se procede a **jugar**. Al finalizar el juego, se calculará la **puntuación** en base al tiempo transcurrido y las vidas perdidas, lo cual se explica en la sección de Puntaje.

Al comenzar el programa, se deberá mostrar la Ventana de inicio, la cual deberá contener un *input* de texto para que el usuario pueda escribir su nombre. Además, la ventana deberá contar con un botón para iniciar el juego y un selector para elegir el archivo base para cargar un mapa **preconstruido**. Al seleccionar la opción de **Jugar**, se deberá validar que el nombre ingresado cumpla con las condiciones establecidas en Ventana de inicio y verificar si se precargó un mapa correctamente o no. En caso de que cumpla con las condiciones, debe cerrarse la Ventana de inicio y abrirse la Ventana de Juego. Si no se seleccione un mapa a cargar, la Ventana de Juego debe abrirse en **modo constructor**. Por el contrario, si se elige cargar un mapa preconstruido, al abrir la Ventana de Juego se deshabilitará el modo constructor y el juego comenzará inmediatamente.

En el modo constructor, la Ventana de Juego deberá mostrar el mapa vacío junto con las estadísticas, opciones de juego y elementos mínimos solicitados detallados en la misma sección. Para personalizar el mapa, se deberá *clickear* el elemento que se desea agregar en el panel de construcción y luego *clickear* la

casilla donde se quiere colocar. Cada casilla puede contener como **máximo** 1 entidad. En caso de que la casilla se encuentre ocupada, se deberá prohibir que otra entidad se posicione en dicha casilla. Una vez pulsado el botón de **Jugar**, se deshabilitará el modo constructor y se procederá a comenzar el juego.

El juego finaliza cuando se cumple alguna de las condiciones que se encuentran detalladas en Fin del juego. Al momento de terminar la partida, la ventana se mostrará totalmente en negro, o bien se esconderán los *sprites* y se indicará si ganaste o perdiste, junto al nombre de usuario y el sonido respectivo de victoria o derrota. Finalmente, se mostrará el botón para **salir** y cerrar el programa.

3. Mecánicas del juego

DCCazafantasmas es un juego que contiene dos modalidades. El primero es el **modo constructor** y el segundo **modo de juego**.

El modo constructor permite crear un mapa de **16 filas** y **11 columnas**, para luego pasar al modo de juego donde comenzará la partida con en el mapa recién creado. Es importante recordar que al momento de crear un mapa, el borde de este siempre estará compuesto por **paredes azules**, por lo que en realidad habrán solo 14 filas y 9 columnas para ubicar elementos.

3.1. Modo constructor

En este modo, el usuario debe ser capaz de construir su propio nivel. Para esto, la interfaz cuenta con un selector de bloques que el usuario puede disponer en el mapa. Cada bloque tiene un número máximo de veces que puede estar presente en el mapa, y este número puede ser distinto para cada tipo de bloque. Queda a tu criterio definir un número para cada tipo de bloque, siempre y cuando los definas como un `PARAMETRO`. No obstante, para **Luigi** y Estrella, ese número **debe ser 1**.

Una vez satisfecho con el mapa creado, el usuario puede presionar el botón de jugar para cambiar de modo. La partida solo puede comenzar si el mapa contiene, como mínimo, a **Luigi** y la Estrella. En otro caso, el juego se mantendrá en el modo constructor.

3.2. Modo de juego

En el modo de juego, el usuario debe conseguir llegar al bloque **Estrella** y presionar la tecla **G**, además de evitar las colisiónes con los fantasmas y bloques que le puedan causar daño.

El personaje principal contará con `CANTIDAD_VIDAS` vidas en el juego. Cuando el personaje pierde una vida, se debe descontar del total de vidas, y volver a la posición inicial hasta que se quede sin vidas. Una vez que no queden vidas el juego debe terminar, **indicando que se perdió la partida por falta de vidas**. Otra forma de finalizar el juego es que se agote el tiempo. En ese caso, **se debe indicar que la derrota es por falta de tiempo**.

3.3. Puntaje

Junto al comienzo del modo de juego, se deberá iniciar una cuenta regresiva de `TIEMPO_CUENTA_REGRESIVA` segundos. Si este tiempo llega a 0 segundos el juego debe terminar y se considera que el usuario perdió la partida.

En caso de terminar la partida satisfactoriamente, el puntaje obtenido se calcula a partir del tiempo restante y la cantidad de vidas ocupadas. La fórmula del puntaje es la siguiente:

$$puntaje\_total = \frac{tiempo\_restante * \texttt{MULTIPLICADOR\_PUNTAJE}}{vidas\_ocupadas}$$

Donde `tiempo_restante` es el tiempo que queda de la cuenta regresiva en **segundos** y `vidas_ocupadas` no puede ser 0, es decir, una vez comienza el juego, se asume que se ocupa 1 vida.

3.4. Comienzo del juego

El comienzo del juego se puede dar en dos casos:

1. Si el usuario selecciona un mapa en la Ventana de inicio, el modo de juego debe comenzar automáticamente, sin posibilidad de pasar por el modo constructor.
2. Si el usuario optó por el modo constructor, el juego debe comenzar solo después que el usuario presiona el botón para **jugar**. Además, el juego solo empezará si en el mapa está posicionado **Luigi** y la Estrella correctamente. Si estas condiciones se cumplen al momento de pulsar el botón para **jugar**, tanto este como el botón de **limpiar** deben quedar inhabilitados y no se podrán presionar nuevamente.

3.5. Fin del juego

El juego debe terminar apenas **cualquiera** de las tres condiciones siguientes ocurran:

1. **Luigi** llega a la **Estrella** donde se encuentra encerrado **Aaossa** y presiona la tecla `'G'` sobre el bloque para liberarlo.
2. La cuenta regresiva de `TIEMPO_CUENTA_REGRESIVA` llega a 0.
3. Un fantasma llega a la misma casilla que Luigi, o Luigi toca la casilla de fuego cuando no le quedan vidas, ~~matándolo~~ convirtiéndolo en fantasma y terminando el juego.

Al terminar el juego se debe reproducir un audio indicando la victoria o derrota de la partida. En el caso de la victoria, se deberá reproducir `stageClear.wav`, mientras que si es una derrota, se deberá reproducir `gameOver.wav`. Además, debe aparecer un mensaje con el nombre del usuario, el resultado del juego y el puntaje final. Finalmente, deberá aparecer un botón de 'salir' para cerrar el programa.

4. Interfaz gráfica

4.1. Modelación del programa

Se evaluará, entre otros, los siguientes aspectos:

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

4.2. Ventanas

Dado que *DCCazafantasmas* 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 tengas una idea de cómo se deberían ver y no es necesario que tu tarea se vea exactamente igual.

4.2.1. Ventana de inicio

Es la **primera ventana** que se debe mostrar al jugador cuando ejecute programa, el cual deberá contener:

- Una **linea de texto editable** para ingresar el nombre de usuario.
- Un **selector** para poder seleccionar y cargar algún mapa existente o bien indicar la opción de mapa vacío.
- Un **botón** para comenzar la partida.
- Un **botón** para salir del programa.

Si el usuario pulsa el botón de comenzar una **nueva partida**, se deberá **verificar** que el nombre de usuario sea **alfanumérico**[1], se encuentre en el rango de `MIN_CARACTERES` y `MAX_CARACTERES` (incluyendo ambos extremos), y que no sea vacío. En el caso de que **no se cumpla alguna restricción**, se deberá notificar mediante un mensaje de error o *pop-up* en la interfaz, indicando el motivo del error. Si **cumple con todo** lo antes mencionado, se deberá cerrar la ventana y se mostrará la Ventana de Juego. Por último, si se selecciona la opción de salir, se deberá cerrar la ventana de inicio y terminar el programa.

El selector deberá contar con `N+1` opciones diferentes, donde `N` corresponde a la cantidad de mapas predefinidos que existen en la carpeta **Mapas**. Este selector deberá mostrar el nombre de cada mapa, el cual corresponde al nombre de cada archivo ubicado en la misma carpeta. Para esto, puedes hacer uso de la librería `os` para obtener esta información. Además, el selector debe incluir una opción para indicar que no se desea cargar un mapa predefinido, vale decir, comenzando con el mapa vacío en el **modo constructor**. El nombre de esta opción debe ser intuitiva para el usuario, por ejemplo, `"mapa vacio"`, `"modo constructor"`, `"no cargar mapa predefinido"`, etc.

La creación de esta ventana puede ser únicamente con Python o apoyarse de QtDesigner. En caso de utilizar esta última herramienta, no olvides subir el archivo `.ui` junto con tu código, o de lo contrario, esta ventana **no será evaluada**.

Figura 2: Ejemplo de Ventana de inicio

4.2.2. Ventana de Juego

Esta ventana se separa en **2 secciones** principales: el **listado de elementos** y **el mapa**. La **primera sección** variará dependiendo del modo en que se encuentre la ventana. **En modo constructor** consiste en el listado de los distintos elementos que pueden ser agregados al mapa y en esta, se encuentra un **selector** donde el usuario puede **filtrar** los elementos en base a su categoría. Las **categorías** son:

[1] Alfanumérico hace referencia a los caracteres formados por **solo** letras (ya sean minúsculas o mayúsculas) sin tildes, **solo** dígitos, o bien letras y dígitos.

1. **Todos**: Cuando esta opción este seleccionada, se deben mostrar todos los elementos.
2. **Bloques**: Cuando esta opción este seleccionada, se deben mostrar solamente los elementos de tipo material, es decir: **Pared**, **Roca**, **Fuego** y **Estrella**.
3. **Entidades**: Cuando esta opción este seleccionada, se deben mostrar solamente los elementos de tipo entidad, es decir: **Luigi**, **Fantasma Vertical** y **Fantasma Horizontal**.

Luego de esto, se encuentra un listado de los distintos **elementos** que el usuario puede agregar al mapa, los cuales deberán cambiar a medida que el usuario seleccione alguna de las opciones mencionadas anteriormente. Además, esta sección presenta **2 botones**: uno para limpiar todos los elementos que se encuentran en el mapa, y otro para comenzar el juego.

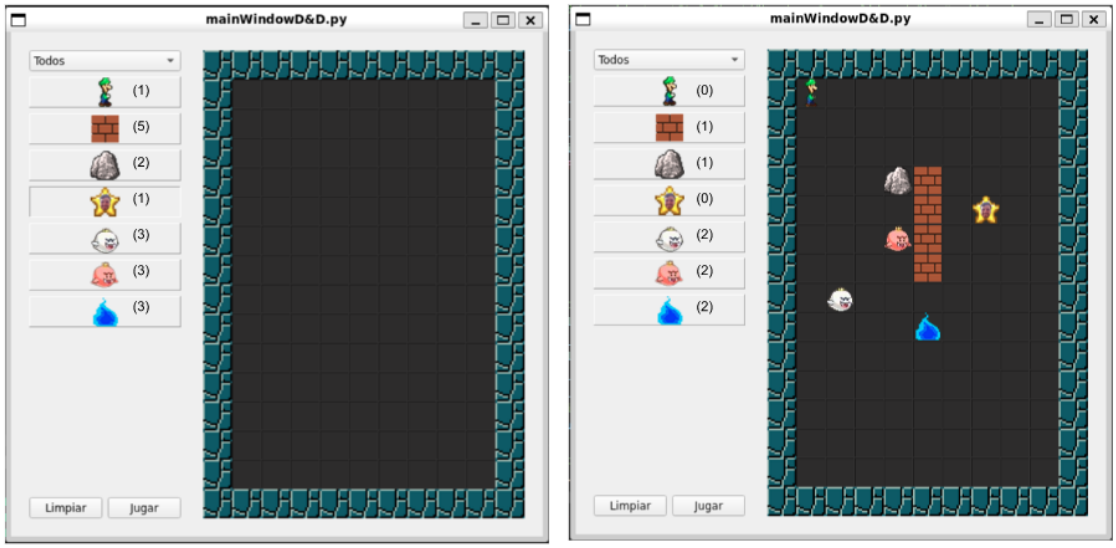
Por otra parte, una vez que el usuario comience a jugar, la ventana cambiará a **modo juego** y deben ocurrir dos mecánicas principales: se deberán deshabilitar o esconder los elementos relacionados al modo constructor mencionados anteriormente y se debrán mostrar los datos relacionados al modo juego (tiempo y vidas) junto con un botón de pausa. Además, ambos botones (**Jugar** y **Limpiar**) deben quedar deshabilitados.

Finalmente, la **segunda sección** consiste en una **grilla** de dimensión **16x11**[2], donde todas las celdas del borde deberán contener **paredes azules**, mientras que el resto deben estar vacías para poder agregar elementos en ellas. Este será el mapa del juego, donde el usuario, mientras se encuentre en el modo **constructor**, podrá agregar tantos elementos como se le sea posible en los lugares que no contengan paredes azules. Una vez que el usuario se encuentre en el modo de **juego**, ya no podrá agregar elementos y comenzará a ejecutarse toda la lógica del juego.

Si el usuario seleccionó algún mapa predefinido en la Ventana de inicio, la **primera sección** sobre la **construcción** del mapa deberá quedar **deshabilitada**, mientras que la **segunda sección** acerca del juego deberá estar completada según las entidades indicadas en el archivo. Por el contrario, si el usuario seleccionó un mapa vacío, se deberá mostrar el mapa sin elementos ubicadas en ella habilitando el **modo constructor**.

La creación de esta ventana debe ser **exclusivamente** desarrollada con Python. **No se revisará esta ventana si fue desarrolla con un `.ui`** (QtDesigner).

[2]Donde 16 corresponde a la cantidad de filas y 11 a la cantidad de columnas.



(A) (B)

Figura 3: (A) Ejemplo de Ventana principal en modo constructor y (B) Ejemplo de Ventana principal con elementos en el mapa

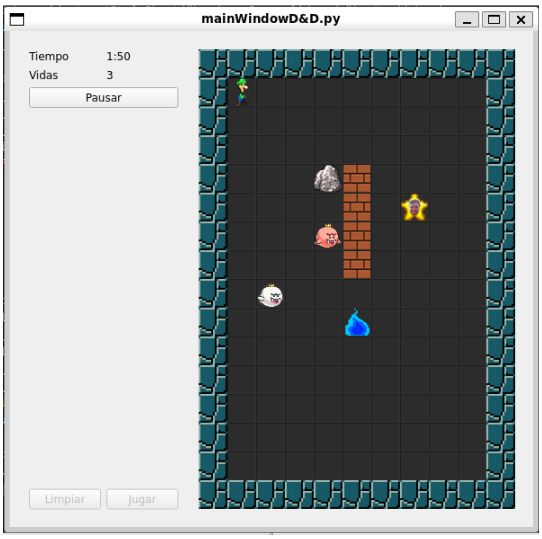


Figura 4: Ejemplo de Ventana principal en el modo Juego

5. Entidades

5.1. Luigi

Luigi es el personaje principal del juego *DCCazafantasmas*, el cual puede ser controlado por el usuario mediante el uso del teclado. Debe existir un único **Luigi** en el juego, es decir, en el modo constructor no se puede insertar más de un **Luigi** al mapa, y el juego no puede partir si no hay un **Luigi** presente en el mapa.

Luigi tiene `CANTIDAD_VIDAS` vidas inicialmente. Si algo le hace daño, **Luigi** perderá una vida y se reiniciará el nivel, haciendo que todos los bloques que se movieron vuelvan a su estado original. Si **Luigi** es herido cuando no le quedan vidas, el juego terminará y se deberá notificar al usuario de la terrible noticia.

El movimiento de este personaje es libre, a menos que tenga una colisión con los límites del mapa o algún otro bloque que se lo impida. Su movimiento es gatillado por el uso de las teclas `WASD`, donde la tecla `W` es moverse hacia arriba, `A` hacia la izquierda, `S` hacia abajo y `D` hacia la derecha. Cuando se presiona una de estas teclas, **Luigi** debe moverse una casilla en la dirección correspondiente.

El desplazamiento de este personaje es **discreto**, pero la animación del movimiento es **continua**, es decir, el personaje se mueve únicamente de una casilla a otra, jamas quedará entremedio de 2 casillas. No obstante, este movimiento debe verse de forma fluida y continua. Para esto, el movimiento entre casillas debe mostrarse como una animación correspondiente a la dirección en que se está moviendo, con una cantidad de **3** ***sprites***, los cuales deben intercalarse entre sí para animar este movimiento entre casillas. Estos *sprites* están especificados en la sección Movimiento de Luigi.

5.2. Fantasma

El fantasma es el principal enemigo del juego. Si el fantasma llega a estar en la misma casilla que **Luigi** cuando a este no le quedan vidas, se terminará el juego y el usuario habrá perdido. En caso contrario, si alcanza a **Luigi** pero aún posee vidas restantes, entonces solo perderá una vida.

La animación del movimiento para el fantasma es idéntica a la de **Luigi**, es decir, se mueve de manera continua con una cantidad de 3 *sprites*, los cuales están especificados en Movimiento de los fantasmas. El movimiento de los fantasmas es independiente al movimiento de **Luigi**, esto significa que cada ciertos segundos, se activa el movimiento de los fantasmas para que se muevan a la siguiente casilla, ocupando la siguiente fórmula:

$$\texttt{tiempo\_movimiento\_fantasmas} = \frac{1}{\texttt{ponderador\_velocidad\_fantasmas}}$$

Donde `ponderador_velocidad_fantasmas` es un número decimal aleatorio entre `MIN_VELOCIDAD` y `MAX_VELOCIDAD`, y `tiempo_movimiento_fantasmas` es la cantidad de **segundos** que deben transcurrir entre un movimiento y otro.

Los fantasmas **NO** colisionan con otros fantasmas, es decir, **SÍ** pueden haber dos fantasmas en una misma casilla. En la siguiente sección se presentan los tipos de fantasmas existentes y se describe su movimiento.

5.2.1. Tipos de Fantasma

- **Fantasma Vertical**

  Este tipo de fantasma solo se puede mover de manera vertical. Si el fantasma choca con un objeto impenetrable, este debe cambiar su dirección y continuar con su movimiento, es decir, si el fantasma se está moviendo hacia arriba y se topa con un objeto impenetrable, el fantasma debe comenzar a moverse hacia abajo hasta encontrarse con otro objeto impenetrable y cambiar de dirección nuevamente. El usuario puede poner hasta un máximo de `MAXIMO_FANTASMAS_VERTICAL` fantasmas de este tipo en el mapa.

- **Fantasma Horizontal**

  El fantasma horizontal sigue la misma lógica que el fantasma vertical pero con un movimiento de horizontal, valga la redundancia. Al igual que el fantasma anterior, si se encuentra con un objeto impenetrable, este debe cambiar de dirección y comenzar a moverse hacia el otro lado. El usuario puede poner hasta un máximo de `MAXIMO_FANTASMAS_HORIZONTAL` fantasmas de este tipo en el mapa.

5.3. Bloques

Todos los bloques deben estar disponibles en el modo constructor para que el usuario los pueda disponer en el mapa. También, todos los bloques presentan algún tipo de interacción con las entidades de personajes, es decir, **Luigi** y los fantasmas.

Existen 3 propiedades de bloques no excluyentes. Un bloque **inamovible** es aquel que tiene las mismas propiedades que los límites del mapa, es decir, al chocar con ellos, los fantasmas deben cambiar de dirección y **Luigi** no es capaz de moverlos. Un bloque **impenetrable** es aquel que cuando es chocado por un fantasma, el fantasma debe cambiar su dirección. Finalmente, un bloque **daño** es aquel que puede quitarle una vida a **Luigi** o purificar a los fantasmas en caso de colisión.

5.3.1. Tipos de Bloques

Dentro del modo constructor, se disponen de 3 tipos de bloques que el usuario tienen a su disposición para incluir en el mapa.

- **Pared**

  Este tipo de bloque es **inamovible e impenetrable**, es decir, ni **Luigi** ni los fantasmas pueden atravesar o mover este bloque. El usuario puede poner hasta un máximo de `MAXIMO_PARED` bloques de este tipo en el mapa.

- **Roca**

  Este tipo de bloque es **impenetrable**, es decir, los fantansmas deben cambiar su dirección si chocan con este bloque. El usuario puede poner hasta un máximo de `MAXIMO_ROCA` bloques de este tipo en el mapa.

  Si **Luigi** se encuentra una casilla adyacente a este bloque y se mueve en la misma dirección, chocando con la roca, entonces la roca deberá moverse a la siguiente casilla en la misma dirección del movimiento. Si la casilla siguiente se encuentra ocupada, la roca no se moverá.

- **Fuego**

  Este bloque es de tipo **daño e inamovible**, es decir, si **Luigi** o los fantasmas se mueven a la casilla donde hay fuego, estos recibirán daño. Para el caso de los fantasmas, estos desaparecerán. Para el caso de **Luigi** colisiona con uno y aún posee vidas restantes, este perderá una y el nivel se reiniciará sin mover el bloque. Por el contrario, si **Luigi** no posee vidas, entonces se terminará el juego. El usuario puede poner hasta un máximo de `MAXIMO_FUEGO` bloques de este tipo en el mapa.

6. Interacción

6.1. Movimiento de los fantasmas

Para el juego, se deberá simular el movimiento continuo de los fantasmas tanto verticales como horizontales. Para esto, se entregaran distintos *sprites* (ver sección de Archivos) los cuales deberán intercalarse entre si para lograr realizar la animación de estos.

(a) Parte 1 (b) Parte 2 (c) Parte 3

Figura 5: Animación de los fantasmas horizontales

(a) Parte 1 (b) Parte 2 (c) Parte 3

Figura 6: Animación de los fantasmas verticales

6.2. Movimiento de Luigi

Al igual que para los fantasmas, el movimiento de Luigi también deberá ser simulado por medio del uso intercalado de *sprites*, dependiendo de la dirección en que vaya el personaje. En el caso de tocar el fuego o algún fantasma, deberá desaparecer y reaparecer en el punto de partida original, además de descontar una vida. En caso de estar quieto, no se deberá modelar ningún tipo de movimiento, vale decir, será un *sprite* estático.

(a) Parte 1 (b) Parte 2 (c) Parte 3

Figura 7: Animación caminata horizontal de Luigi

(a) Parte 1 (b) Parte 2 (c) Parte 3

Figura 8: Animación caminata de Luigi hacia arriba

(a) Parte 1

(b) Parte 2

(c) Parte 3

Figura 9: Animación caminata de Luigi hacia abajo

6.3. *Click*

La interacción con los botones del juego se deberá hacer por medio de *clicks*. Además de los botones, podrás interactuar con otros objetos presentes en el juego.

6.3.1. Constructor

Si se decide utilizar el modo constructor en lugar de cargar un mapa existente, el jugador podrá diseñar un nuevo mapa mientras se encuentre en el modo contructor. Para poder agregar una nueva entidad en el mapa, se deberán seguir los siguientes pasos:

1. Hacer *click* sobre la entidad del panel de construcción que se desea agregar.
2. Hacer *click* sobre la casilla del mapa en donde se desea agregar la entidad.
3. Revisar si la casilla seleccionada es válida, vale decir, que no esté ocupada por otra entidad. En caso de ser inválida, se deberá notificar al jugador **en la misma ventana**, no mediante consola.

Si el usuario desea limpiar el mapa, deberá hacer *click* en el botón de **Limpiar**. **No es necesario implementar alguna mecánica para eliminar 1 elemento del mapa a la vez**, pero si deseas realizarlo por comodidad, puedes especificarlo en el *Readme*.

6.4. *Cheatcodes*

Para esta tarea, y con el fin de ~~facilitar la corrección~~ mejorar la jugabilidad, se deberá poder presionar de manera seguida o simultanea ciertas combinaciones de teclas durante la partida (según tu preferencia[3]), las cuales deberán llevar a cabo ciertas acciones:

- K + I + L: esta combinación de teclas elimina a todos los villanos del mapa.
- I + N + F: esta combinación de letras congela el contador de la partida, dando tiempo indefinido para terminarla. También otorga vidas infinitas a **Luigi**, es decir, si **Luigi** es herido, el nivel se resetea, pero no se restan vidas. En este caso, el puntaje calculado a partir de la fórmula de Puntaje sería el máximo posible, vale decir, el máximo tiempo restante junto al mínimo de vidas ocupadas.

6.5. Pausa

En la ventana del juego, deberá existir un botón de pausa para detener o reanudar el juego, el cual también deberá activarse pulsando la tecla **P**. Al pausar el juego, se deberá detener el contandor del tiempo transcurrido, además de cualquier otro tipo de movimiento, incluyendo a los villanos e imposibilitando al personaje **Luigi** de moverse. En caso de volver a hacer *click* sobre el botón de pausa o volver a presionar la tecla P, el juego y todos sus elementos se deberán reanudar.

[3]Se debe referenciar en el *readme* el método usado, ya sea pulsar teclas en orden o de forma simultánea.

6.6. Tomar Estrella

Para poder ganar la partida, el jugador deberá llegar hasta la **Estrella** y pulsar la tecla **G** para liberar al mago **Aaossa**. Luego de esto, el juego dará por finalizado, ganando la partida.

7. Archivos

Para el desarrollo de la tarea, deberás hacer uso de los siguientes archivos entregados:

- *Sprites*: Corresponden a los elementos visuales que se encuentran en la carpeta `sprites`.
- Sonidos: Corresponde a los sonidos utilizados al ganar y perder el juego, explicados en Fin del juego. Estos archivos están incluidos en la carpeta `sounds`.
- Mapas: Corresponde a diferentes mapas predefinidos que podrás seleccionar en la Ventana de inicio. Puedes encontrarlo en la carpeta `mapas`.

Adicionalmente, tu programa debe ser capaz de leer los siguientes archivos:

- `parametros.py`: En esta tarea se entregará este archivo con los parámetros del ancho y largo de la grilla y deberás agregarle todos los parámetros en `ESTE_FORMATO` señalados a lo largo del enunciado, además de cualquier otro que veas pertinente añadir, como los `PATHS` o rutas de archivos.

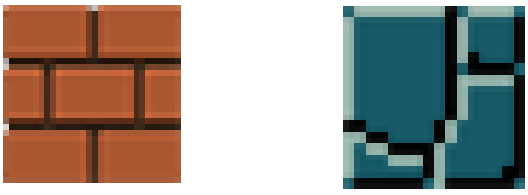
  Dani *ojo que este archivo aún no existe, para que recuerden subirlo.*

7.1. *Sprites*

Esta carpeta contiene todas las subcarpetas de las diferentes imágenes en formato `.png` que se ocuparán para tu tarea, entre ellos se encuentra:

- **`Elementos`**: Esta carpeta consta con los cuatro elementos necesarios para construir el mapa, además del logo del juego.

(a) Fuego (b) Estrella (c) Piedra



(d) Pared (e) Pared azul

Figura 10: Elementos a utilizar en la construcción del mapa.

- **`Personajes`**: Esta carpeta contiene los 3 personajes a utilizar para desarrollar la tarea.

Figura 11: Caminata de Luigi

Figura 12: Avance del fantasma horizontal

Figura 13: Avance del fantasma vertical

- **`Fondos`**: Esta carpeta contiene el fondo del menú de inicio.

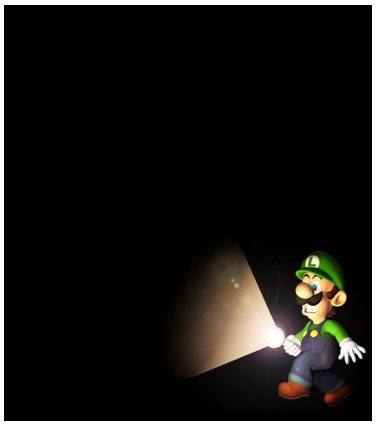


Figura 14: *Fondo menú de inicio*

7.2. Sonidos

Esta carpeta contendrá los sonidos que pueden implementarse en el juego, todo detallado en la sección de Fin del juego. Se encuentran `gameOver.wav` que se utilizará cuando el jugador pierda y `stageClear.wav` para cuando se gane el juego.

7.3. Mapas

Esta carpeta contendrá todos los mapas predefinidos para jugar *DCCazafantasmas*. El formato de cada mapa será una secuencia de 9 caracteres por línea y en total serán 14 líneas. Dando un total de 9x14. Esto es porque el borde del mapa siempre estará compuesto por paredes azules, así que estos no son guardados en el mapa, sino que incluidos automáticamente al momento de cargar el mapa.

La simbología de cada entidad en el mapa es la siguiente:

- `"-"`: casilla vacía.
- `"L"`: casilla con **Luigi**.
- `"P"`: casilla con pared.
- `"F"`: casilla con fuego.
- `"H"`: casilla con fantasma de movimiento horizontal.
- `"V"`: casilla con fantasma de movimiento vertical.
- `"S"`: casilla con la estrella.
- `"R"`: casilla con una roca.

Puedes asumir que estos mapas siempre estarán correctamente diseñados, es decir, siempre habrá un **Luigi** en el mapa y una estrella, y nunca habrá un símbolo que no corresponde a los indicados anteriormente.

A continuación se muestra un mapa generado a partir de un archivo.

```
L--------
---------
---------
---RP----
----P-S--
---VP----
----P----
-H-------
----F----
---------
---------
---------
---------
---------
```

Figura 15: Ejemplo del mapa generado por el archivo anterior

7.4. `parametros.py`

En este archivo se encontrarán el ancho de la grilla como `ANCHO_GRILLA` y el largo de la grilla como `LARGO_GRILLA`, los cuales **no debes modificar**. Además, deberás agregar todos los parámetros mencionados anteriormente en el enunciado en `ESTE_FORMATO`, en donde cada línea almacena una constante con su respectivo valor. En tu tarea deberás **importar**[4] correctamente este archivo y utilizar los parámetros almacenados.

[4]Para mas información revisar el material de modularización de la semana 0.

Además de incluir los parámetros descritos anteriormente, es importante incluir todo tipo de valor que será constante en tu programa, como los *paths* de archivos que se utilizarán y todo valor constante creado por ti. Es importante que los nombres de los parámetros sean declarativos, de lo contrario se caerá en una mala práctica.

Este archivo debe ser **importado como módulo** y así usar sus valores almacenados. En caso de que el enunciado no especifique un valor de algunos de los parámetros, deberás asignarlo a tu criterio, procurando que no dificulte la interacción con el juego y considerando que los ayudantes podrán modificarlo para poder corroborar que el juego está implementado de forma correcta.

Por otro lado, parámetros predeterminados como la ruta de los archivos o las dimensiones de la ventana y sus elementos nunca serán modificados, de tal manera que no interfiera con el funcionamiento del juego.

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

8.1. Volver a jugar (2 décimas)

Al acabar el juego y mostrar el resultado, debe dársele la opción al usuario de poder reiniciar el nivel. Esto implica jugar de nuevo el mapa, no así entrar en modo constructor. La forma de dar la opción queda a criterio del estudiante (ya sea un botón, una nueva ventana, u otros).

8.2. *Follower Villain* (3 décimas)

Follower Villain es un tercer tipo de fantasma el cual en vez de moverse solamente en direcciones horizontal o vertical, debe seguir a Luigi, avanzando una casilla cada `TIEMPO_MOVIMIENTO_FANTASMAS`. Este fantasma debe evitar chocar con obstáculos[6], pero pueden morir si se encuentra con una casilla de fuego. El movimiento de este fantasma se reduce a las casillas inmediatamente colindantes (arriba, abajo, izquierda, derecha), pero **no en diagonal.**

Para optar este *bonus*, el fantasma debe ser una entidad adicional en el modo constructor y debe funcionar tal cómo se explicó en el párrafo anterior.

8.3. *Drag and Drop* (3 décimas)

Al seleccionar una entidad del panel de construcción, en vez de posicionarla en el mapa mediante el uso del *click* izquierdo, deberás usar *drag and drop* para arrastrar el bloque a su posición deseada en el mapa. Al soltar el bloque, este debe quedar en una posición válida, es decir, no puede quedar por encima de otro elemento, o fuera del área permitida. En caso de que se intente lo anterior, no se debe permitir colocar el bloque y se deberá notificar al jugador que la posición no es válida. Una vez se suelte el bloque en una posición válida, la entidad se sumará al mapa en la casilla correspondiente.

[5] Esta nota es sin considerar posibles descuentos.

[6] Se recomienda utilizar una lógica sencilla para evitar el atoramiento con los limites del mapa y el bloque Piedra

9. Propuesta de avance

Para esta tarea no habrá una entrega de avances, sin embargo se propone uno con el fin de que comiencen implementando una de las partes importantes de la tarea. El avance corresponderá a manejar apropiadamente **un tipo de colisión** que funcione en todas las direcciones, realizar el **movimiento fluido** de **Luigi** y el de los fantasmas.

Para ello, puedes crear una ventana en donde se encuentre una Pared y **Luigi**. **Luigi** deberá colisionar con la pared, impidiendo que siga avanzando. Además, puedes implementar el movimiento de los fantasmas y lograr que se superpongan entre sí, como dicta el enunciado.

Para este avance podrás pedir un *feedback* general de lo implementado, durante la sala de ayuda para esta tarea.

10. `.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.pdf
- Carpeta de *Sprites* entregada junto al enunciado.
- Carpeta de Sonidos entregada junto al enunciado.
- Carpeta de Mapas entregada junto al enunciado.

Recuerda **no ignorar tu archivo de parámetros y archivos** `.ui`**, 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.

11. Entregas atrasadas

Entregas con atraso tendrán una penalización en la calificación máxima obtenida acorde a la política de atrasos. Sin embargo, tendrás a disposición cupones para eliminar días de atrasos.

Política de atraso:

- Hasta 1 día de atraso (00:01 a 24:00 hrs de atraso) tendrán calificación máxima 6,0.
- Hasta 2 días de atraso (24:01 a 48:00 hrs de atraso) tendrán calificación máxima 4,0.
- Después de las 48 hrs, no se aceptarán entregas atrasadas y se aplicará calificación mínima 1,0.

Cupones:

- Durante el semestre cada estudiante dispondrá de 2 cupones.
- Cada cupón permite eliminar 1 día de atraso en la tarea recién entregada.

Posterior a la fecha de entrega de la tarea se abrirá un formulario de ***Google Form***, en el cual deberán indicar si desean optar por una entrega atrasada y si harán uso de 0, 1 o 2 cupones para la entrega.

12. 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.

En 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 Coordinador de Bienestar o al ayudante de Bienestar de tu sección, puedes hacerlo escribiendoles a sus respectivos **correos**.

13. 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 3.10.
- 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 prohibida. 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. El no incluir este archivo o bien se encuentra vacío conllevaría un **descuento** en tu nota.
- Entregas con atraso entre 0 a 24 horas tendrán como calificación máxima 6,0. Entregas entre 24 y 48 horas tendrán calificación máxima 4,0. Más allá de este plazo, la calificación será mínima (1,0).
- Tendrás a tu disposición solo **dos** cupones en el semestre. Cada cupón permite eliminar 1 día de atraso en la tarea recién entregada.
- En ambas situaciones, ya sea entregar atrasado y/o utilizar 0, 1 o 2, deberán rellenar el **formulario** correspondiente que será liberado una vez terminado el plazo oficial de la tarea.
- 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).