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

IIC2233 Programación Avanzada (2019-2)

Tarea 02

Entrega

- **Avance de tarea**
  - **Fecha y hora:** miércoles 2 de octubre de 2019, 20:00
  - **Lugar:** GitHub — Carpeta: `Tareas/T02/`
- **Tarea**
  - **Fecha y hora:** sábado 12 de octubre de 2019, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T02/`
- `README.md`
  - **Fecha y hora:** lunes 14 de octubre de 2019, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T02/`

Objetivos

- Utilizar conceptos de interfaces y PyQt5 para implementar una aplicación gráfica e interactiva.
- Entender y aplicar los conceptos de *back-end* y *front-end*.
- Implementar una separación entre el *back-end* y el *front-end*.
- Aplicar conocimientos de *threading* en interfaces.
- Aplicar conocimientos de señales.

Índice

1. *DCCampo*

Luego de los exitosos resultados con *Initial P*, la paz volvió al DCC. Sin embargo, ha aparecido un nuevo enemigo: el malvado **Emperador EnZurg**, hermano gemelo malvado de **Enzo**, quien ha tomado el control absoluto y ha desterrado a nuestro amigo Enzo a las lejanas tierras del *DCCampo*.

A pesar de que pueda sonar como una terrible situación, es un lugar bastante agradable y tranquilo, donde puedes realizar diversas actividades como cultivar alimentos, talar árboles, entre otros. Tu misión es crear un programa con PyQt5 que ayude a Enzo a pasar sus días en estas tierras y le permita juntar suficientes recursos para volver al DCC.

Figura 1: Logo de *DCCampo*.

2. Flujo del juego

DCCampo es un juego de simulación, en donde el jugador tomará el rol de **Enzo** en su aventura por tener una granja exitosa para así juntar dinero y volver al DCC con orgullo.

Al iniciar el programa se debe mostrar la Ventana de Inicio, en donde el jugador debe indicar el archivo de granja a cargar. Si el archivo es inválido, se debe avisar en pantalla. En caso contrario, se procede a la Ventana de Juego, que es en donde se realizará toda la partida.

El juego consiste en lograr mantener tu granja activa por la mayor cantidad de días posibles, mediante acciones la agricultura y venta de recursos. Además de contar el paso de los días, este también posee un reloj interno que mide las horas de cada día, lo que controla la ocurrencia de ciertos eventos.

DCCampo se desarrolla mediante una interfaz gráfica, con Mapas que representan la granja y sus elementos y un personaje posicionado sobre este, controlado por el Jugador, además de una barra lateral donde es posible ver información relevante para el jugador en todo momento.

El jugador comienza la partida con una granja completamente abandonada y una cantidad inicial de `MONEDAS_INICIALES`[1], las que pueden ser utilizadas para comprar semillas, herramientas y **El Gran *ticket* de regreso al DCC** en la **Tienda**. También posee un inventario portátil, donde almacena todas sus pertenencias.

En el mapa aparecerán diversos recursos: leña, oro y cultivos, los cuales se podrán **recolectar** por medio del movimiento del personaje. Una vez compradas ciertas herramientas para el jugador, le será posible: **cortar árboles** para obtener leña; o **arar tierra** y así volverla disponible para plantar semillas en el mapa. Una vez plantada una semilla, esta tardará un tiempo determinado en crecer y producir cultivos.

Es importante notar que Enzo posee una **barra de energía**, la cual indica la energía disponible que tiene para el día. Las acciones descritas anteriormente tienen asociado un gasto de energía para el jugador. Esta

[1] Los elementos escritos en `ESTE_FORMATO` son parámetros que tendrás que escribir e importar desde el archivo `parametros_generales.py`.

energía puede restaurarse nuevamente al **ir a dormir y pasar al siguiente día** o **ingresar un código secreto**.

Finalmente, el juego puede terminar bajo dos condiciones: se logran juntar las monedas necesarias y comprar **El Gran *ticket* de regreso al DCC**, significando la victoria del jugador, o **Enzo** agota su barra de energía de forma completa, lo que implica la derrota del jugador.

3. Entidades

3.1. Jugador

El personaje principal, **Enzo**, es controlado por el jugador. Enzo puede moverse libremente por la granja a través de las teclas `WASD`, cuidando que no atraviese obstáculos como **árboles** y **piedras**. Tampoco debe atravesar el **borde del mapa**.

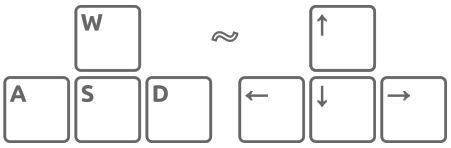

Figura 2: Movimiento `WASD`

Figura 3: Estados de movimiento del jugador

Es importante mencionar que el movimiento del personaje debe ser **continuo**, esto quiere decir, que se mueve píxel por píxel. El personaje debe avanzar una cantidad fija de píxeles cada vez que el usuario aprieta una tecla, esta cantidad está definida por un parámetro `VEL_MOVIMIENTO`.

El personaje tiene un valor de energía máxima que inicialmente está definido por el parámetro `ENERGIA_JUGADOR`. Dicha energía disminuye cada vez que el jugador **tala árboles** o **ara la tierra**. Para recuperar energía el jugador debe dormir. La cantidad de energía que recupera está especificada por el parámetro `ENERGIA_DORMIR`. Cuando se acaba la energía del jugador, este se desmaya y pierde el juego.

La energía del jugador debe estar representada por una **barra de energía**, la cual se debe actualizar cada vez que la energía aumenta o disminuye. La barra debe encontrarse en la ventana lateral del juego.

Finalmente, el personaje es capaz de interactuar con los elementos de la granja, ya sea, **caminado** sobre los recursos para recolectarlos o **situándose** sobre la casa y la tienda para activar sus respectivos efectos. Todo lo anterior se explica con mayor profundidad en Interacción del usuario con *DCCampo*.

3.2. Cultivos

Los cultivos son fundamentales en la vida; los podemos encontrar en todas partes, nos proveen de alimentos y recursos que necesitamos. Las tierras del *DCCampo* no son una excepción.

Para conseguir un cultivo, deberás comprar una semilla en la tienda, luego dejarla en un **espacio de cultivo** (tierra arada) que no esté ocupado. Cada cultivo tiene varias **fases de crecimiento**. Para pasar de una fase a otra deberá haber pasado una cantidad determinada de tiempo en el juego. La cantidad de fases y el tiempo que tarda un cultivo en crecer una fase vendrá dado por el archivo `parametros_plantas.py`. Cada fase debe poder diferenciarse visualmente.

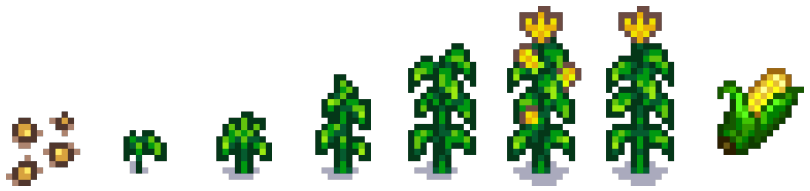

Figura 4: Distintas fases de un cultivo de maíz.

Cuando un cultivo haya llegado a su última fase, proporcionara **frutos**. Las plantas pueden ser cosechadas de dos modos, dependiendo si es permanente o no.

Si la **planta es permanente**, luego de ser cosechada vuelve a la penúltima fase de crecimiento, por lo que se podrá volver a cosechar cuando crezca; en el caso de que la **planta no sea permanente**, entonces una vez cosechada, el terreno donde se plantó vuelve a convertirse en tierra arada.

Las plantas que podrás encontrar son:

- **Alcachofa**: es una planta de crecimiento **no permanente**.
- **Choclo**: es una planta de crecimiento **permanente**.

Las semillas de estas plantas pueden ser compradas en la Tienda. Para obtener más información sobre las plantas revisa el archivo `parametros_plantas.py`.

3.3. Herramientas

Son aquellos elementos que permiten al personaje **Enzo** realizar ciertas acciones en la granja. Las herramientas presenten en el programa son:

- **Azada:** se utiliza para arar la tierra; con ella puedes transformar un **espacio libre** en un **espacio cultivable**.
- **Hacha:** se utiliza para poder obtener **leña** de **árboles**.

El jugador no contará con ninguna de ellas al iniciar una partida, por lo que no será capaz de realizar esas acciones inmediatamente. Sin embargo, puede obtenerlas comprándolas en la tienda una vez que junte suficiente dinero, dándole acceso a la acción correspondiente.

3.4. Aparición espontánea

Las entidades de aparición espontánea son aquellas que tienen cierta probabilidad de aparecer en el mapa cada vez que se inicia un nuevo día en el *DCCampo*. Ambas entidades aparecen de la misma manera: al pasar cada día, con cierta probabilidad, puede aparecer uno (y solo uno) de cada tipo de entidad, y de hacerlo, aparece en cualquier celda disponible del mapa. Existen dos tipos de entidades espontáneas:

- **Árboles:** su probabilidad de aparición está definida por el parámetro `PROB_ARBOL`. De ellos puedes obtener **leña**, un recurso que puedes vender para obtener dinero. Para recolectar la **leña**, Enzo deberá tener un **hacha** y hacer *click* sobre el árbol; entonces, se despejará el espacio del árbol y aparecerá leña en su lugar.
- **Oro:** su probabilidad de aparición está definida por el parámetro `PROB_ORO`. Para recolectarlo el personaje debe desplazarse sobre él.

Debes cuidar que no aparezca más de un elemento espontáneo por espacio. En caso de que el resultado sea equiprobable entre dos elementos, deberás **dar prioridad al árbol**.En caso de que no queden espacios libres disponibles, tampoco deben aparecer nuevos elementos.

3.5. Recursos

Como ya sabes, el jugador debe ayudar a **Enzo** a comprar **El Gran *ticket* de regreso al DCC**, y así volver al DCC. Para esto, este es capaz de recolectar recursos y luego venderlos en la tienda. Para ser recolectados, el personaje debe caminar sobre ellos, y así se agregan al inventario. Los recursos que puedes recolectar son los siguientes:

- **Leña**: la leña solo aparece al talar árboles, en la misma posición dónde se encontraba el árbol. La leña estará disponible para la recolección durante una cantidad fija de tiempo en segundos, almacenado en `DURACION_LENA`, y luego desaparecerá.
- **Oro**: el oro también estará disponible para la recolección durante una cantidad fija de tiempo en segundos, almacenado en `DURACION_ORO`, y luego desaparecerá.
- **Frutos**: los frutos se cosechan de los cultivos una vez que han crecido hasta su última fase. Los frutos que puedes recolectar son choclo y alcachofas. Una vez que los frutos están listos, el jugador puede recolectarlos. Es importante notar que a diferencia de los demás recursos, los frutos **no** desaparecen con el tiempo.

El precio de venta de todos los recursos se encuentra en el archivo `parametros_precios.py`.

4. Interfaz

4.1. Modelación del programa

Para esta tarea deberás crear un programa utilizando `PyQt5` y *threads*[2], donde se evaluará:

- La correcta moduralización del programa, para así obtener una **adecuada separación entre *back-end* y *front-end***.
- El **correcto uso de señales** para modelar todas las acciones de la interfaz: movimientos, interacción, acceso, corroboración, etc.
- La **ausencia de dependencias circulares** entre los elementos que compongan tu tarea.

El juego debe poseer un **reloj interno** que controle el flujo del tiempo, es decir, las horas del día y así el progreso de ciertos eventos: crecimiento de las plantas, el pasar de los días, etc. La rapidez con la que pasa el tiempo (cuántos segundos dura un día, por ejemplo) queda a tu criterio y debes definirlo como un parámetro.

4.2. Ventanas

El campo es un lugar muy amplio y complejo. Es por esto que deberás implementar **varias ventanas** que representen las partes más importantes de este. Tu programa deberá implementar **como mínimo** las ventanas que se describirán a continuación. Cabe destacar que, a menos de que se explicite en cada subsección, cualquier decisión relacionada con el diseño de estas ventanas (distribución dentro de la

[2]Esto incluye `QThreads`, `QTimer`, etc.

ventana o número de ventanas en que se puede dividir, por ejemplo) **queda a tu libre criterio**. Todas las imágenes son ejemplos visuales a modo de referente, pero no deben ser interpretados como el resultado final esperado.

4.2.1. Ventana de Inicio

Esta ventana es la que debe aparecer al ejecutar el programa, en donde se pregunta por el **nombre del archivo** que contiene al mapa a cargar para la partida, **incluyendo la extensión de este**.

Cuando el jugador ingrese un nombre de archivo, el programa revisará si dentro de la carpeta `mapas` se encuentra un archivo con el nombre ingresado, y en caso contrario mostrará un mensaje de **error**, no permitiendo el acceso al juego hasta que el nombre sea válido. Una vez ingresado un nombre de archivo correcto, esta ventana se **cierra** y se abre la **Ventana de Juego** en una ventana distinta.

Figura 5: Ejemplo de ventana de inicio

4.2.2. Ventana de Juego

En esta ventana es donde se desarrolla el juego en sí. Debe estar separada en tres áreas, donde una representa el **mapa** y al **personaje**. Como sabemos, este último puede interactuar con elementos del mapa ya sea **situandose** sobre ellas (en el caso de la **casa** y la **tienda**) o **caminando** sobre ellas (en el caso de los recursos) y el usuario puede generar acciones sobre el con el teclado y haciendo *click*. Otra área representa representa el **inventario** del jugador, mientras que la última área reprenta a la **ventana lateral**, donde en todo momento se muestran *stats* relevantes del juego que son sujetos a cambios en tiempo real. Los elementos mínimos a presentar en dicha barra son:

- Número de días jugados.
- Hora actual.
- Cantidad de energía restante de Enzo, **representada por una barra de progreso**[3].
- Dinero actual de Enzo.
- Botón para **salir** de la partida.
- Botón para **pausar** el juego.

[3] Para eso, puedes investigar sobre el *widget* `QProgressBar`.

Finalmente, la forma de implementar gráficamente el **inventario** del jugador queda a tu discreción, mientras sea **intuitivo** y **sencillo** de utilizar. A continuación se presentan algunos ejemplos de cómo podría ser implementada la ventana de juego:

Figura 6: Ejemplo de Juego con dos ventanas

Figura 7: Ejemplo de Juego con una ventana

4.2.3. Tienda

La tienda es una ventana donde el jugador adquiere nuevos elementos para su vida en la granja, además de vender los que ya no le son útiles. Dentro de la tienda se debe presentar de forma individual cada producto en venta, junto a su precio. En caso de que el jugador no posea el dinero suficiente para comprar el producto, se le debe **notificar mediante la interfaz**.

Nuevamente, el cómo implementar la tienda queda a tu criterio. A continuación se presenta una posible implementación:

Figura 8: Ejemplo de tienda

5. Interacción del usuario con *DCCampo*

A lo largo del juego, el usuario puede interactuar de diferentes formas con *DCCampo*. Las maneras especificas en que deben implementarse estas interacciones se detallan a continuación:

- ☐ Para realizar ciertas acciones el jugador debe hacer un ***drag and drop*** desde el inventario hasta el mapa. Los elementos que debes poder arrastrar son:
  - **Semillas**: para plantar un cultivo, deberá ser posible arrastrar las semillas hacia la parte del mapa donde se desean plantar. Las semillas solo se pueden plantar en casillas del tipo **espacio cultivable** desocupadas, es decir, espacios libres que ya han sido arados y que no tienen ningún otro cultivo hasta el momento.
- ☐ Para realizar algunas acciones el jugador solo deberá hacer ***click***. Si se hace *click* sobre:
  - Un espacio libre, este se transformará en un **espacio cultivable**. Recuerda que esta acción solo está disponible si el jugador cuenta con la **Azada** en su inventario, y pierde energia al hacerlo.
  - Un árbol, este se transforma en leña en la misma ubicación. Recuerda que esta acción solo está disponible si el jugador cuenta con el **Hacha** en su inventario, y pierde energia al hacerlo.
  - Para poder comprar y vender dentro de la **Tienda**, se debe poder hacer *click* sobre distintos elementos para vender o comprar una unidad. Cómo queda a tu criterio.

- ☐ Para poder acceder a ciertas funcionalidades el personaje deberá **pararse** sobre los siguientes elementos del mapa:
  - Casa: al caminar sobre ella, permite a Enzo recargar su energía e iniciar un nuevo día de forma automática.
  - Tienda: al caminar sobre ella, se abre la ventana de la tienda, que permite al jugador comprar o vender objetos.
- ☐ Para **recoger** los siguientes elementos, el personaje deberá **caminar** por sobre ellos:
  - Cultivos ya cosechados de la planta.
  - Leña ya cortada de un árbol.
  - Oro que aparezca en el mapa.
- ☐ En el caso de que el personaje intente caminar por sobre alguno de los siguientes objetos deberá **chocar** con ellos, impidiéndole el paso:
  - Un árbol que aún no ha sido talado.
  - Una piedra.
  - El borde del mapa[4].

El **plantar una semilla**, **talar un árbol**, **arar un espacio de tierra**, **cosechar una planta** o **recoger un recurso** son acciones que consumen energía del personaje. La cantidad de energía que cuesta realizar cada acción está detallada en el archivo `parametros_acciones.py`.

6. Archivos

Para lograr simular la granja de manera correcta deberás hacer uso de los siguientes archivos:

- `mapa_1.txt` y `mapa_2.txt`, los cuales son explicados en la Subsección 6.1: Mapas.
- `parametros_plantas.py`, `parametros_recursos.py` y `parametros_acciones.py`, explicados en la Subsección 6.3: Módulos entregados.

Además deberás crear el archivo `parametros_generales.py`, el cual se debe rellenar con cualquier parámetro que consideres correcto, los parámetros mínimos pedidos están explicados en la Subsección 6.2: `parametros_generales.py`.

6.1. Mapas

Para implementar las ventanas en donde se realiza el juego es necesario que cargues el mapa del juego, a través de archivos que describen la distribución de los elementos de este. La carpeta `mapas` contiene dos archivos: `mapa_1.txt` y `mapa_2.txt`. Estos archivos codifican mediante una matriz de letras y espacios la distribución espacial de la ventana en donde interactúa tu personaje.

Cada elemento de los mapas tiene el mismo tamaño, y están delimitados por un cuadrado de $N \times N$ píxeles, donde N es una constante que debes definir en `parametros_generales.py`. Cada mapa será rectangular y tendrá dimensiones $i \times j$ elementos, las cuales estarán representadas en cada archivo.

[4] Es decir, el personaje no puede salir del mapa.

Se definen **cinco tipos de espacios** que debes considerar al armar tu juego:

- **Espacio libre `(O)`:** espacios en donde el personaje puede desplazarse libremente y que puede arar para transformarlo en un espacio cultivable. Al comenzar un nuevo día pueden aparecer **árboles** u **oro**, más detalle de esto en Aparición espontánea.
- **Espacio cultivable `(C)`:** espacio en el cual se pueden plantar semillas para obtener plantas y vegetales, más detalles en Cultivos. El personaje puede desplazarse encima de estos espacios libremente. Al inicio del juego no habrán plantas en este espacio.
- **Piedra `(R)`:** espacios por los cuales el personaje no puede desplazarse. No puede volverse un espacio cultivable, ni tampoco aparecer elementos de aparición espontánea.
- **Casa `(H)`:** espacios ocupados por el hogar del jugador, este estará constituido siempre por agrupaciones de 2×2 celdas. El personaje puede posicionarse sobre estos espacios. Aquí no aparecerán elementos espontáneos, ni se puede arar esta celda.
- **Tienda `(T)`:** Espacios ocupados por la tienda del *DCCampo*. Al igual que la casa, será un conjunto de 2×2 celdas. Aquí no aparecen elementos espontáneos, ni se puede convertir en espacios cultivables.

Para efectos del juego debes cargar uno de los archivos entregados, cuya ruta debes cargar a partir del archivo obtenido en la Ventana de Inicio. Tu programa debe ser capaz de leer cualquiera de estos archivos (u otro que siga el mismo formato) y funcionar de manera independiente de las características que presenten y su tamaño.

```
O O R O O O O O O O O O O O O O O O O O
O C C C C C C O O O O H H O O C C C C O
O C C C C C C O R O O H H O O C C C C O
O C C C C C C O O O O O O O R O C C C C O
O O R O O R O O O O O O O O O O R O O O O
O O T T O O O O O O O O O O O R O O O O O
O O T T O O O C C C C C C C O O O O O O O
O O O O O O O C C C C C C C O O O O O R O
O O O O O O O C C C C C C C O O O R O O O
O R O O R O O O O O O O O R R O O O O O O
```

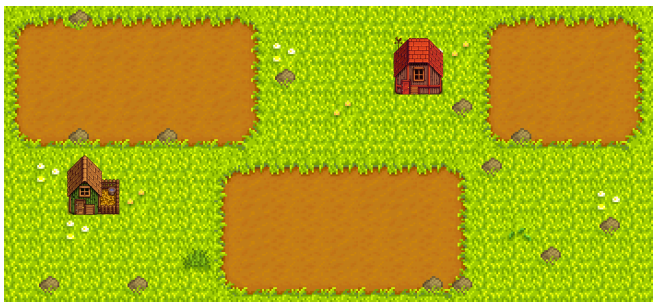

Figura 9: Ejemplo de un archivo que contiene información para cargar el predio y su visualización

6.2. `parametros_generales.py`

Deberás crear y rellenar un archivo `parametros_generales.py` que contenga toda la información de los parámetros[5] y números mencionados anteriormente, al igual que los *paths* de los archivos que utilice tu programa y cualquier otra constante que consideres adecuada. Cada línea de este archivo almacena una constante con su valor respectivo. Es importante recalcar que debes **importar** este archivo **como un módulo** y así usar los valores almacenados.

6.3. Módulos entregados

Los siguientes módulos representan los costos y valores de distintos parámetros base del programa. Estos archivos te los entregamos y debes **importarlos**, pero no debes agregarles parámetros adicionales ni subirlos a tu repositorio.

[5] Los parámetros indicados en el enunciado son los elementos escritos en `ESTE_FORMATO`.

6.3.1. parametros_plantas.py

El archivo `parametros_plantas.py` contiene información de las plantas existentes en el juego. La información será entregada de la siguiente manera:

- `PRECIO_SEMILLA_XXXX`: precio de la semilla de XXXX en la tienda.
- `TIEMPO_XXXX`: tiempo en segundos que tarda el cultivo XXXX en crecer una fase.
- `FASES_XXXX`: número de fases de crecimiento que tiene el cultivo XXXX.
- `COSECHA_XXXX`: número de recursos de XXXX que entrega la planta XXXX al ser cosechada.
- `PATH_SPRITES_XXXX`: `str` que muestra el *path* de la carpeta con los *sprites* del cultivo XXXX.

Los *sprites* de los cultivos tendrán de nombre `stage_K`, siendo `K` la etapa de crecimiento de la planta. Si `K` es igual al valor `Fases` del cultivo, entonces se puede cosechar.

Los valores `XXXX` mostrados anteriormente pueden ser `ALCACHOFAS` o `CHOCLOS` para el archivo `parametros_plantas.py`.

6.3.2. parametros_precios.py

El archivo `precio_precios.py` contiene los precios de todos los objetos que se pueden comprar y vender en el juego. La información será entregada de la siguiente manera:

`PRECIO_XXXX = ZZZZ`

Siendo `XXXX` el nombre del objeto y `ZZZZ` su precio.

6.3.3. parametros_acciones.py

El archivo `parametros_acciones.py` contiene los costos de energía de todas las acciones existentes en el juego. La información será entregada de la siguiente manera:

- `ENERGIA_HERRAMIENTA:` cuanta energía pierde Enzo al ocupar una herramienta.
- `ENERGIA_COSECHAR:` cuanta energía pierde Enzo al cosechar un cultivo.
- `ENERGIA_RECOGER:` cuanta energía pierde Enzo al recoger un recurso del suelo.

7. Funcionalidades extras

7.1. Combinaciones de teclas

Con la finalidad de ~~facilitar la corrección~~ mejorar la experiencia de los jugadores al jugar *DCCampo* se te pide implementar las siguientes combinaciones de teclas de forma que al apretarse **todas** las teclas al mismo tiempo[6] se realice el efecto deseado:

- `K + I + P`: al presionarse estas tres teclas al mismo tiempo se debe restaurar toda la energía de Enzo, sin la necesidad de dormir.

[6] *Tip:* para esto se te recomienda utilizar un *set* para almacenar las teclas que actualmente están siendo presionadas. Cada vez que se realice un `keyPressEvent` agregas la tecla al *set* y cada vez que se haga `keyReleaseEvent` se elimina del *set*.

- `M + N + Y`: al presionarse estas tres teclas al mismo tiempo aumenta en `DINERO_TRAMPA` el dinero total de Enzo.

7.2. Pausa

Para poder pausar y después continuar el juego, se pide que implementes un botón de pausa. Cada vez que la pausa se encuentre activada, se deberá apreciar visualmente, es decir, se deberá detener la animación de los elementos del juego y no deberá aumentar el tiempo del reloj interno.

8. *Sprites*

Para esta tarea, se te entregaran *sprites*[7] para representar gráficamente las distintas entidades del juego. Éstas estarán contenidas en la carpeta `sprites` la cual contiene las carpetas:

- `personaje`: contiene las *sprites* de movimiento del personaje.
- `cultivos`: contiene las imágenes de las distintas etapas de los cultivos, sus semillas, sus frutos y demases. Las *sprites* están agrupadas en directorios según la planta a la que corresponden: `alcachofa` y `choclo`.
- `recursos`: contiene las *sprites* de los distintos recursos que se pueden obtener durante la partida.
- `mapa`: contiene las imágenes necesarias para construir el mapa del juego, es decir, las representaciones de las celdas mencionadas en Mapas.
- `otros`: contiene una variación de *sprites* que no corresponde a ninguna de las carpetas anteriores, como los árboles y el dinero.

En algunas carpetas hay imágenes adicionales que te podrán ser de utilidad para tu tarea.

En caso de que quieras hacer uso de tus propias *sprites* **deberás guardarlas en carpetas con otro nombre e indicarlo en tu README.md**. Para obtener las *sprites* de los *spritesheets* se te aconseja utilizar las herramientas *ShoeBox* o *ezgif.com* la cual te permite cortar, quitar el fondo y renombrar las *sprites*.

Como se indicará más adelante en la sección `.gitignore`, si usas los *sprites* que te entregamos no debes subir la carpeta *sprites* a tu repositorio, por lo que **no debes cambiar el nombre de las imágenes**. Para facilitarte el acceso a las direcciones de las imágenes (que puedes acceder de forma más lógica) te recomendamos utilizar un diccionario como el que se muestra a continuación.

```
celdas_mapa = {
    'espacio_libre': 'sprites/mapa/tile006',
    'espacio_cultivable': 'sprites/mapa/tile061',
}
```

9. *Bonus*

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

[7] Para más información puedes visitar el siguiente ***link***.

1. La nota en tu tarea (sin *bonus*) debe ser **igual o superior a 4.0**[8]
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 **7 décimas**.

9.1. Pesca (3 décimas)

Uno de los ***hobbys*** que Enzo quiere aprender en su nueva vida de granjero es la pesca, por esta razón deberás permitirle lanzar el anzuelo al agua e intentar conseguir pescados de la siguiente manera:

Figura 10: Ejemplo del minijuego de pesca

Figura 11: Ejemplo del minijuego terminado con el jugador victorioso

Debes agregar un botón en la ventana de juego que permita a Enzo ir a pescar, luego de apretar el botón se mostrará una interfaz parecida a la anterior, la cual deberá poseer un rectángulo grande de juego (el azul en el ejemplo), un rectángulo pequeño de éxito (el verde en el ejemplo) y un rectángulo de pescado (el pescado en el ejemplo)[9].

El pescado siempre comenzará lo más a la izquierda posible y el rectángulo verde estará siempre lo más a la derecha. El largo del rectángulo verde debe estar definido por `LARGO_RECTANGULO` que es un porcentaje del largo del rectángulo azul. En el ejemplo, el valor de `LARGO_RECTANGULO` es 7/24, ya que el rectángulo verde ocupa 7/24 del espacio que ocupa el rectángulo azul.

El pescado deberá tener un largo de `LARGO_PESCADO`, el cual al igual que `LARGO_RECTANGULO` es un porcentaje del largo del rectángulo azul.

El pescado se deberá mover hacia la derecha a una velocidad de `VEL_PESCADO`, parámetro cuyo funcionamiento queda a tu discreción. Cada X cantidad de segundos, siendo X un valor aleatorio entre 0 y `TIEMPO_PESCADO`, el pescado debe cambiar su sentido para moverse al lado contrario. Si se está moviendo a la izquierda moverse a la derecha y si se está moviendo a la derecha se deberá mover a la izquierda.

Mientras el pescado se mueve el usuario debe tener la opción de apretar la barra de espacio, la cual detendrá al pez y terminará el minijuego. Si el minijuego termina con el rectángulo del pescado completamente dentro del rectángulo de éxito (rectángulo verde), entonces el jugador gana dinero equivalente a `PRECIO_PESCADO`. Luego el juego debe volver a como estaba antes de que el jugador comenzara a pescar. En caso de que no ocurra lo anterior, el minijuego termina en derrota y se cierra la ventana del minijuego.

9.2. Casa (4 décimas)

Cansado de dormir en la intemperie, Enzo decide construir una casa de verdad. Pero como Enzo el granjero solo vive dentro de tu programa, es tu deber construírsela.

Debes crear un nuevo mapa llamado `mapa_casa.txt`, donde vendrá el diseño del interior de la casa de Enzo. Y tu programa debe ser capaz de detectar cuando Enzo camine sobre alguna casilla de `Casa` **(H)** en el mapa, cuando esto ocurra se deberá **recargar el mapa de la Ventana de Juego**. El nuevo mapa deberá representar el interior de la casa de Enzo, en esta Enzo podrá moverse y recargar su energía.

[8] Esta nota es sin considerar posibles descuentos.
[9] Revisar los siguientes *links* para ejemplos del minijuego: **minijuego**, **gif**.

La mínimas condiciones que debe tener la casa son:

- Tener al menos una casilla de puerta, la cual al ser pisada por el jugador lo transporte de vuelta a la ventana de juego anterior, es decir, el mapa de la granja.
- Tener al menos una casilla de cama, en la cual el personaje pueda cargar su energía al posicionarse sobre ella.
- Respetar los límites de la casa, es decir, el personaje no debe poder trasladarse por espacios que no esten contenidos en `mapa_casa.txt`.

El formato que utilices para `mapa_casa.txt` y la carpeta con los *sprites* que uses para el piso, la cama y la puerta de la casa deberán estar explicados en el archivo `README.md` de tu tarea.

10. Avance de tarea

Para esta tarea, el avance corresponderá a implementar:

- El cargado visual del mapa a partir de los archivos entregados.
- El *drag and drop* de algún cultivo hasta el mapa ya cargado.

Es importante que al realizar estas implementaciones, el cultivo solo se pueda soltar en alguna posición válida, es decir, una celda de cultivo que se encuentre disponible. En caso de que se suelte en una posición inválida, se deberá cancelar el traslado y devolver la imagen del cultivo al punto inicial. Para el avance, solo es necesario que la imagen de un choclo se pueda arrastrar; éste no necesita crecer a medida que pase el tiempo de las fases. Tampoco es necesario implementar gráficamente el inventario.

A partir de los avances entregados, se les brindará un *feedback* general de cómo de lo que implementaron en sus programas y además, les permitirá optar por **hasta 2 décimas** adicionales en la nota final de su tarea.

11. `.gitignore`

Para esta tarea **la carpeta llamada `sprites` deberá ser ignorada** con un `.gitignore` que deberá estar dentro de tu carpeta `T02/`. Puedes encontrar un ejemplo de `.gitignore` en el siguiente ***link***.

En caso de que hagas uso de tus propias *sprites*, deberás guardarlas en otra carpeta y asegurarte que no sean ignoradas por el `.gitignore`.

Tampoco deberás subir el enunciado, los archivos contenidos en la carpeta `mapas` y los archivos `.py` entregados: `parametros_plantas.py`, `parametros_precios.py` y `parametros_acciones.py`.

Se espera que no se suban archivos autogenerados por programas como PyCharm, o los generados por 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.

12. 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.

13. Importante: Corrección de la tarea

Para esta tarea, el carácter funcional del juego 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.

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, se recomienda el uso de *prints* para ver los estados del sistema (etapas de las plantas, energía del personaje, etc.) pero no se evaluará ningún ítem por consola. Esto implica que hacer *print* del resultado una función, método o atributo no valida, en la corrección, que dicha función esté correctamente implementada. Todo debe verse reflejado en la interfaz. Por ejemplo, si se utiliza la energía del personaje en una actividad, que hagan *print* de su energía disminuyendo no será válido, para este ítem se evaluará que la barra de energía en la interfaz disminuya.

14. 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).