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

IIC2233 Programación Avanzada (2022-1)

Tarea 2

Entrega

- **Avance de tarea**
  - **Fecha y hora:** miércoles 11 de mayo de 2022, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- **Tarea**
  - **Fecha y hora:** domingo 22 de mayo de 2022, 20:00
  - **Lugar:** Repositorio personal de GitHub — Carpeta: `Tareas/T2/`
- `README.md`
  - **Fecha y hora:** domingo 22 de mayo de 2022, 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. *DCComando espacial*

Luego de una semana en la cual estás tranquilamente programando el *DCCasino* comienzas a escuchar un altercado afuera de tu hogar, te asomas por la ventana y al momento de abrirla aparece *Terminator-Dog* con una misión para ti. Él te informa que viene del año 2050 en donde la resistencia lo ha creado y enviado desde el *DCComando espacial* para entrenar la *IA* encargada de las defensas contra los *aliens* que conquistaron el planeta tierra.

Luego de contarle que eras un experto programador, a *Terminator-Dog* se le ocurrió que con tus conocimientos de *threading* e interfaces gráficas le podrías ayudar a entrenar estas defensas y evitar que el planeta sea conquistado. Debido a esto, te encomienda la tarea de diseñar un programa que les permita practicar la puntería de las defensas eliminando a las naves invasoras y así proteger el futuro el planeta.

Figura 1: Logo de *DCComando espacial*

2. Flujo del programa

DCComando espacial es un juego que consiste en ayudar a *Terminator-Dog* a eliminar los diversos *aliens* que intentan llegar al planeta tierra para conquistarlo. Para esto, tendrás a disposición un **arma** con la que deberás entrenar la IA a lo largo de la galaxia para que reconozca a los enemigos y aumente su automatización y puntería. El armamento contará con cierta cantidad de **munición** limitada, por lo que tendrás que acabar con tus objetivos con la menor cantidad de balas y en el menor tiempo posible para asegurar el máximo puntaje.

El programa contará con distintos **escenarios** para jugar: Tutorial Lunar, Entrenamiento en Júpiter e Invasión Intergaláctica. Cada uno de estos escenarios tiene sus propios *aliens* y dificultad asociada. Además, los escenarios contarán con niveles ilimitados que deberás ir superando para avanzar de un nivel a otro. Al momento de seleccionar un escenario, deberás acabar con todos tus objetivos en el menor tiempo posible. En caso de lograrlo, se calculará un puntaje en base a la cantidad de *aliens* eliminandos, tiempo y balas restantes, la dificultad del escenario y el número del nivel en el cual se jugó. Finalmente, tendrás la opción de avanzar al siguiente nivel del mismo escenario o salir.

Al iniciar el programa se mostrará la **ventana de inicio**, en la que el jugador podrá seleccionar la opción de abrir la **ventana de rankings** para observar los mejores puntajes registrados, o la opción de **jugar**. Si se elige la opción de iniciar el juego, se deberá cerrar la ventana de inicio y abrir la **ventana principal**. En esta, podrás seleccionar el **escenario** en el que deseas participar y entregar tu nombre de usuario. Una vez ingresados, se dará paso a la **ventana de juego**, la cual mostrará el mapa, las estadísticas y opciones de juego. Al comenzar el nivel, los *aliens* comenzarán con la invasión y deberás cazarlos a todos. El nivel terminará una vez hayas eliminado a todos los invasores, o te hayas quedado sin munición o tiempo.

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

3. Mecánicas del juego

DCComando espacial 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. Arma

La interacción del usuario con el programa se realizará mediante la **arma** del juego. Esta se debe usar para disparar a los *aliens* del nivel seleccionado en un tiempo determinado dependiendo de la dificultad del nivel.

Para ello, al iniciar el juego la **mira** debe localizarse al **medio de la ventana**, vale decir, el punto medio de los ejes `(x, y)` dependiendo de las dimensiones de la ventana. Durante el juego, la mira podrá moverse libremente en 4 direcciones (arriba, abajo, derecha e izquierda) y disparar para destruir a los *aliens*. Cada vez que se **dispare** con el arma, el **color** de la mira debe cambiar de negro a **rojo** durante `1` segundo y deberá reproducirse el **sonido de disparo**. El movimiento de la mira y el efecto de sonido explicará en mayor detalle en la sección Interacción con el usuario.

El arma contará con una cantidad **limitada** de balas al inicio del nivel, y estará determinada por el doble de la cantidad total de *aliens* del nivel:

$$cantidad\_balas = cantidad\_alien\_nivel \cdot 2$$

Ten en cuenta que se debe gastar una bala con cada disparo y las balas sobrantes al terminar el nivel **no se acumulan** para el siguiente, por lo que deberán **reiniciarse** al comenzar un nuevo nivel.

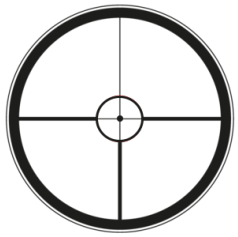
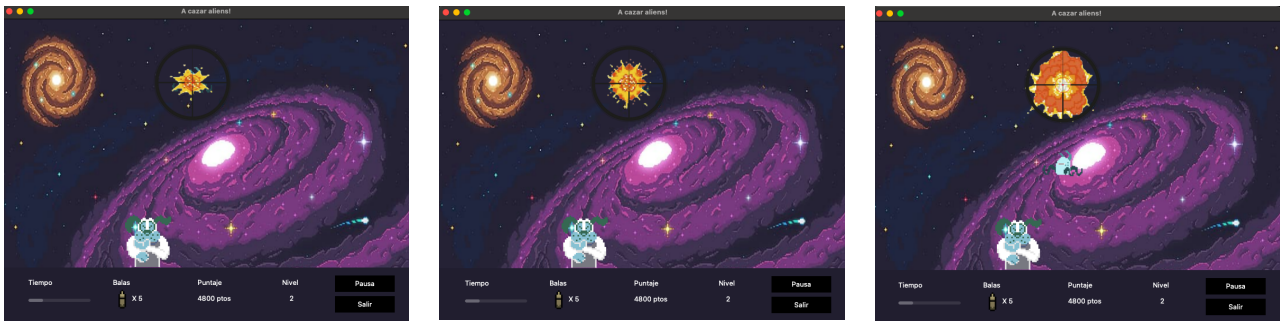


Figura 2: Ejemplo de *sprite* del arma

3.2. *Aliens*

Para detener la invasión debes destruir a todos los ***aliens***. Cada uno de ellos debe aparecer en una **posición aleatoria** dentro del mapa (sin sobrepasar los límites de la ventana) y moverse en **diagonal**. En el caso de colisionar con algún borde deberá **rebotar** y cambiar la dirección actual. Este movimiento se explicará con mayor profundidad en la sección Interacción con el usuario.

Al dispararle a un *alien*, este debe destruirse mostrando una pequeña animación, de manera que se muestre un efecto de explosión y desaparecer de la ventana.



(a) Disparo fase 1 (b) Disparo fase 2 (c) Disparo fase 3

Figura 3: Animación destrucción de *alien*

Los *aliens* deben tener una velocidad inicial `VELOCIDAD_ALIEN`, que corresponde a un vector (x, y) que contiene los pixeles por unidad de tiempo que avanza el *alien* en el eje x y eje y, respectivamente. Sin embargo, dependiendo del escenario en el que se encuentre, estas constantes deben multiplicarse por un ponderador correspondiente a la dificultad de dicho escenario, esto se explica en mayor detalle en la subsección Escenarios y Dificultad.

La cantidad total de *aliens* de un nivel estará determinada por el doble del número del nivel:

$$cantidad\_alien\_nivel = nivel \cdot 2$$

Sin embargo, no deben aparecer todos estos *aliens* al mismo tiempo, sino deben ir apareciendo en parejas de manera que al destruir a ambos *aliens*, aparezca la siguiente pareja.

(a) *sprite* de *alien lunar*

(b) *sprite* de *alien lunar* destruido

(c) *sprite* de *alien* de Júpiter

(d) *sprite* de *alien* de Júpiter destruido

(e) *sprite* de *alien* galáctico

(f) *sprite* de *alien* galáctico destruido

Figura 4: Ejemplo de *sprites* de *aliens*

3.3. *Terminator-Dog*

Además del arma y los *aliens*, existe un personaje amigo llamado *Terminator-Dog*, que te acompañará en la lucha contra los enemigos. Este personaje debe estar presente durante tu partida, sin movimiento. Sin embargo, al final de cada nivel, el *sprite* debe cambiar al *Terminator-Dog* con el *alien* correspondiente en la mano, por un total de `TIEMPO_TERMINATOR_DOG` segundos. Por el contrario, si se **pierde** el nivel, no se

mostrará el personaje con este *sprite*. Una vez termine el tiempo se dará por terminado el nivel y se dará paso al Fin del nivel.

3.4. Escenarios y Dificultad

Antes de comenzar el juego podrás escoger entre 3 **escenarios** distintos para jugar: Tutorial lunar, Entrenamiento en Júpiter e Invasión intergaláctica. En cada uno de ellos habrá distintos tipos de *aliens* y dificultades.

Al iniciar el juego, el nivel tendrá una duración de `DURACION_NIVEL_INICIAL` segundos y los *aliens* se moverán con una velocidad `VELOCIDAD_ALIEN` pixeles por unidad de tiempo (recordar que corresponde a un vector $(velocidad\_x,\ velocidad\_y)$) teniendo en consideración el **ponderador**. Cada vez que se pase al siguiente nivel, el tiempo de este disminuiría y la velocidad de los *aliens* aumentará. Dicha variación estará dada por la dificultad del escenario en el que se esté jugando.

3.4.1. Tutorial Lunar

Este escenario es el más **fácil**, perfecto para jugadores principiantes. Tendrá un ponderador de dificultad `PONDERADOR_TUTORIAL` con un valor entre `0.9` y `1` que afectará a la velocidad de los *aliens* y el tiempo del nivel:

$$velocidad\_alien\_tutorial = \frac{velocidad\_alien\_nivel\_anterior}{\texttt{PONDERADOR\_TUTORIAL}}$$

$$duracion\_nivel = duracion\_nivel\_anterior \cdot \texttt{PONDERADOR\_TUTORIAL}$$

3.4.2. Entrenamiento en Júpiter

Este escenario es de dificultad **intermedia**, perfecto para jugadores que buscan mejorar sus habilidades de destrucción extraterrestre. Tendrá un ponderador de dificultad `PONDERADOR_ENTRENAMIENTO` con un valor entre `0.8` y `0.9` que afectará a la velocidad de los *aliens* y el tiempo del nivel:

$$velocidad\_alien\_entrenamiento = \frac{velocidad\_alien\_nivel\_anterior}{\texttt{PONDERADOR\_ENTRENAMIENTO}}$$

$$duracion\_nivel = duracion\_nivel\_anterior \cdot \texttt{PONDERADOR\_ENTRENAMIENTO}$$

3.4.3. Invasión intergaláctica

Este escenario es de dificultad **extrema**, solo apto para los mejores jugadores listos para salvar el planeta. Tendrá un ponderador de dificultad `PONDERADOR_INVASION` con un valor entre `0.7` y `0.8` que afectará a la velocidad de los *aliens* y el tiempo del nivel:

$$velocidad\_alien\_entrenamiento = \frac{velocidad\_alien\_nivel\_anterior}{\texttt{PONDERADOR\_INVASION}}$$

$$duracion\_nivel = duracion\_nivel\_anterior \cdot \texttt{PONDERADOR\_INVASION}$$

Importante : Los ponderadores se deberán aplicar incluso desde el **nivel 1**. Esto implica que en caso de cursar el primer nivel, la velocidad y duración de los *aliens* estarán dados mediante las siguientes fórmulas:

$$velocidad\_alien\_inicial = \frac{\texttt{VELOCIDAD\_ALIEN}}{ponderador\_escenario}$$

$$duracion\_nivel = \texttt{DURACION\_NIVEL\_INICIAL} \cdot ponderador\_escenario$$

En donde:

- *ponderador_escenario*: corresponde al ponderador del escenario seleccionado.

3.5. Puntaje

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

$$puntaje\_nivel = \texttt{int}(\frac{cantidad\_alien \times 100 + (tiempo\_restante \times 30 + balas\_restantes \times 70) \times nivel}{dificultad})$$

En donde:

- *cantidad_alien*: son la cantidad de *aliens* en el nivel.
- *tiempo_restante*: es el tiempo que queda del nivel.
- *balas_restantes*: son las balas que no fueron utilizadas al terminar el nivel.
- *nivel*: es el número de nivel en el que se está jugando[1].
- *dificultad*: es el ponderador correspondiente al escenario en el que se está jugando.

3.6. Fin del nivel

Una vez que se destruya a **todos** los *aliens*, se acaben las balas o se acabe el tiempo, el nivel finalizará y se mostrará la **ventana post-nivel** con las estadísticas del juego. Solo si se gana el nivel (si destruye a todos los *aliens*), podrás pasar al siguiente.

3.7. Fin del juego

Si se acaban las balas o se termina el tiempo antes de destruir a todos los *aliens* en la **ventana post-nivel** se deberá notificar la derrota del jugador. Luego, el jugador tendrá la opción de volver a la **ventana de inicio** para iniciar una nueva partida. El nombre de usuario y su puntaje **acumulado** se deberán almacenar automáticamente en el archivo `puntajes.txt` una vez que se **presione** el botón para volver al menú principal.

En el caso de que el jugador decida ponerle fin al juego sin haber perdido el nivel, puede presionar el botón para volver al menú principal, y su usuario y puntaje acumulado será almacenado en el archivo `puntajes.txt`.

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***, y un **diseño cohesivo** de **bajo acoplamiento**
- Correcto uso de **señales** y ***threading*** para modelar todas las interacciones en la interfaz.
- Presentación de la información y funcionalidades pedidas (puntajes o balas, 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.

[1] El nivel inicial parte en **1**

4.2. Ventanas

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

4.3. Ventana de inicio

Es la primera ventana que se debe mostrar al jugador al momento de ejecutar el programa. Debe incluir un botón para comenzar una nueva partida y un botón para ver el *ranking* de jugadores. Si el usuario aprieta el botón de **jugar**, se debe cerrar la ventana y mostrar la ventana principal. De otro modo, si el jugador selecciona la opción de **ver el *ranking*** de puntajes, se deberá mostrar la **ventana de *ranking***.

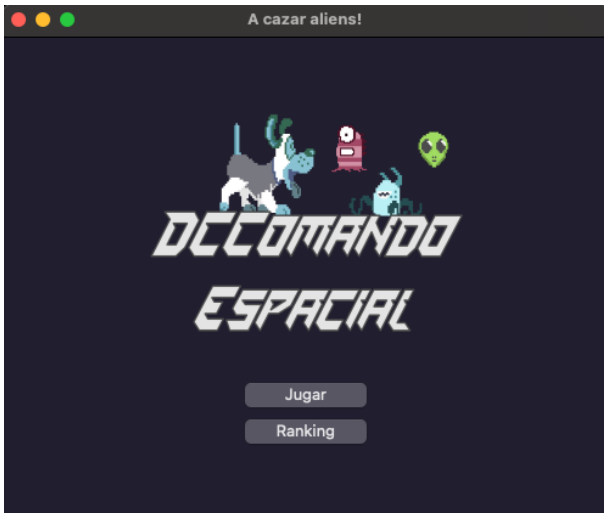


Figura 5: Ejemplo de ventana de inicio de *DCComando espacial*

4.4. Ventana de *ranking*

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

Figura 6: Ejemplo de ventana de ranking de *DCComando espacial*

4.5. Ventana principal

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 el ambiente según la **dificultad** del juego y el tipo de alien que tendrá que cazar. También, debe tener una línea de texto editable para ingresar el **nombre de usuario**. Si el jugador desea **iniciar una partida nueva**, se deberá verificar si el nombre de usuario cumple la restricción **alfanumérica** y que **no sea vacío**. En el caso de que no cumpla, se deberá notificar mediante un **mensaje** de error o *pop-up*. Una vez seleccionado el ambiente y creado el nombre de usuario, se deberá cerrar la ventana y dar paso a la **ventana de juego** para comenzar el nivel.

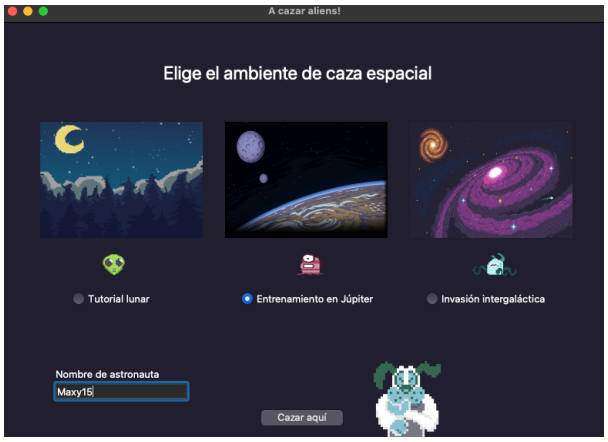


Figura 7: Ejemplo de ventana principal de *DCComando espacial*

4.6. Ventana de juego

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

- Tiempo restante del nivel

- Balas restantes del nivel
- *Aliens* destruidos en el nivel
- *Aliens* restantes para completar el nivel
- Puntaje acumulado sin contar el nivel
- Nivel actual

Además, tu ventana debe mostrar el ambiente elegido en la **ventana principal**, el alien del respectivo ambiente y la mira para apuntar y dispararle. La mira debe **cambiar de color negro a rojo** cada vez que dispares y también, se debe mostrar una animación de *Terminator-Dog* riéndose una vez finalizado el nivel.

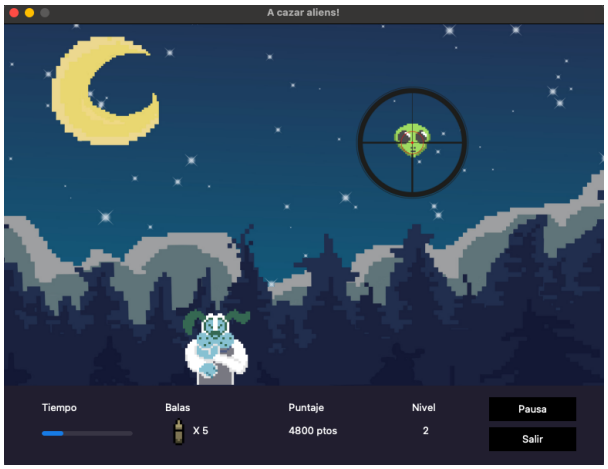


Figura 8: Ejemplo de ventana de juego

El nivel será completado cuando destruyas a todos los *aliens* del nivel, en donde debe aparecer una animación de *Terminator-Dog* mostrando el resultado de la cacería. Finalmente, si el jugador pierde una vez terminado el tiempo o las balas o completa el nivel, este nivel se terminará y se dará paso a la **ventana de post-nivel**. En caso de que el jugador decida salir a la **venta de inicio**, se guardará el puntaje total obtenido de todos los niveles y deberá registrarse en el archivo `puntajes.txt` siguiendo su formato.

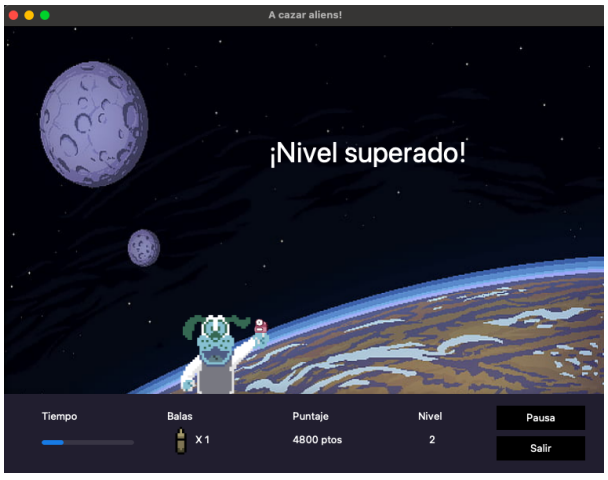


Figura 9: Ejemplo de ventana de juego con el nivel superado

4.7. Ventana de post-nivel

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

- Nivel actual
- Balas restantes
- Tiempo restante
- Puntaje obtenido en el nivel
- Puntaje total acumulado
- Botones para continuar partida y salir del juego

Si el jugador superó el nivel, se deberá notificar con un mensaje en pantalla y se deberán mostrar habilitada la opción para continuar jugando el siguiente nivel. De lo contrario, se deberá notificar al usuario que no puede continuar jugando y se tendrá deshabilitado el botón mencionado anteriormente. Luego, si se selecciona la opción de continuar partida, se deberá abrir la ventana de juego para comenzar el siguiente nivel.

Figura 10: Ejemplo de ventana de post-nivel

5. Interacción con el usuario

5.1. Movimiento del disparador

En la interfaz del juego, poseerás total control de la mira del arma con la que dispararás a los *aliens* que vayan apareciendo. Por esto, se necesitará precisión y fluidez en el movimiento del arma. Se deben usar los `sprites` mostrados a continuación:

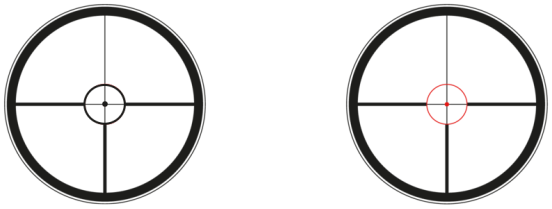


Figura 11: Cambios de disparador en movimiento y en ataque

El `sprite` izquierdo de la figura debe ser utilizado en todo el movimiento que involucre la puntería en el juego, mientras que el de la derecha debe usarse **únicamente** al disparar. El cambio al disparador rojo debe durar exactamente `1` segundo, acompañado del sonido del disparo mencionado en la sección Sonidos.

El movimiento debe realizarse únicamente con las teclas `WASD`, por lo que se debe apreciar una fluidez considerable al mover la mira por todas las direcciones de la pantalla. Por otro lado, el disparo debe efectuarse mediante la tecla `espacio`[2]. En caso de mantener pulsada esta tecla por más tiempo, se deben efectuar todos los disparos correspondientes (aunque eso implique perderlos todos), además de mantener el `sprite` del disparador rojo el tiempo equivalente a la tecla presionada.

5.2. Movimiento de los Aliens

Si bien el usuario no tendrá control sobre el movimiento de los *aliens* que aparecerán en cada nivel de la partida, igualmente juega un papel importante a la hora de ser perseguidos por la mira y reaccionar al disparo. La aparición de alienígenas en cada ronda se efectua según la siguiente fórmula:

$$cantidad\_aliens = nivel \cdot 2$$

Esto quiere decir que si estamos en el nivel 1, independiente del escenario seleccionado, deben aparecer 2 *aliens*. En caso de ir en el nivel 2, deberán aparecer 4 *aliens* en total durante el nivel. Por otro lado, si en el nivel 2 deben aparecer 4 *aliens* en total, **siempre en pantalla deben haber un máximo de 2 alienígenas**. Por ejemplo, si estamos en el nivel 2 y se deben disparar a los 4 *aliens*, primero se mostrará solo un par de ellos. Luego de haberles disparados a ambos, aparecerán los otros dos. El nivel terminará una vez eliminado la *cantidad_aliens* correspondiente.

Es importante saber que la aparición de los *aliens* debe ser **aleatoria** en la pantalla, pudiendo ser en cualquier parte del escenario, exceptuando el panel de estadísticas inferior. Tampoco deberán **superponer** un *alien* sobre otro en su aparición, por lo que deberás verificar que ambos cuerpos no se solapen apenas se muestren inicialmente en la ventana. Sin embargo, independiente de la ubicación del *alien*, su movimiento siempre se debe regir según las características mencionadas a través del vector de velocidad.

5.2.1. Acción de rebotar

Mientras los *aliens* se encuentren volando, su movimiento debe estar dado por `VELOCIDAD_ALIEN` como vector `(x, y)`. Cada vez que el *alien* choca con el límite de la ventana debe rebotar hacia otra dirección, por lo que nunca se podrá tener un movimiento que dependa sólo de un eje. Este **rebote** consiste en el cambio de dirección contraria al chocar con algún borde de la pantalla.

Importante: Para realizar un cambio de dirección, basta con multiplicar por **-1** una de las coordenadas `(x,y)`. Por ejemplo, si el aliens va en dirección diagonal hacia la esquina superior derecha y choca con el límite derecha de la ventana, se debe multiplicar la coordenada `x` por **-1**, de tal manera que el `sprite` siga subiendo pero esta vez con dirección a la izquierda.

[2]En caso de tener problemas con esta tecla, se puede utilizar otra **siempre y cuando** se mencione en el `README.md`

- *Ejemplo 1*: Si la velocidad del *alien* está dada por (2, 3) y se rebota con la pared de la derecha (eje `y`), se debe multiplicar por **`-1`** la coordenada `x`, dando como resultado el vector (-2, 3).
- *Ejemplo 2*: Si la velocidad del *alien* está dada por (-1, 2) y se rebota con la pared de arriba (eje `x`), se debe multiplicar por **`-1`** la coordenada `y`, dando como resultado el vector (-1, -2).

Por otra parte, cada vez que un *alien* reciba un disparo, su `sprite` debe cambiar al señalado en la sección *Aliens*. Este cambio de `sprite` debe ser instantáneo al presionar la tecla `espacio`. Junto con el cambio de `sprite` del *alien*, se debe agregar una secuencia de `sprites` de explosión, la cual estará en la misma posición compartida entre el *alien* y el disparo. Esta secuencia debe comenzar desde que se realiza el disparo. A continuación se muestra un ejemplo de las fases de la explosión:

(a) Disparo fase 1 (b) Disparo fase 2 (c) Disparo fase 3

Figura 12: Secuencia de sprites de la explosión

5.3. Movimiento del *Terminator-Dog*

Al finalizar el nivel una vez se hayan disparado a todos los *aliens*, se deberá mostrar una pequeña animación de *Terminator-Dog* riéndose y mostrando el *alien* del escenario destruido por pocos segundos. En caso de que en el nivel se haya perdido, ya sea por falta de munición o por tiempo, esta animación no se mostrará y se deberá pasar a la **ventana post-nivel**.

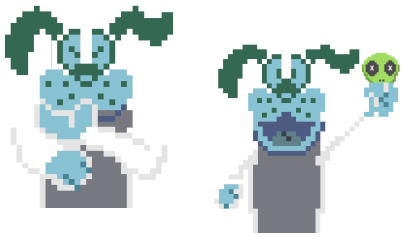


Figura 13: Aparición *Terminator-Dog* en la Ventana de juego

5.4. *Click*

Cada vez que se quiera interactuar con algún botón u opción de la interfaz de una ventana, debe ser mediante un *click* en ellas.

5.5. *Cheatcodes*

Con la única finalidad de ~~facilitar la corrección~~ mejorar la experiencia en cada nivel de una partida del juego, es posible presionar ciertas combinaciones de teclas de forma continuas o simultáneas (según tu preferencia[3]) durante la partida:

- **O + V + N + I**: Esta combinación otorga balas infinitas **exclusivamente** por el nivel en el que te encuentras.

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

- **C + I + A**: Esta combinación pasa inmediatamente al siguiente nivel de la partida.

5.6. 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 el arma y se pausarán las demás mecánicas del juego, considerando el movimiento de *aliens* y el tiempo. Una vez que se vuelva a presionar la tecla **P** o se haga *click* sobre el botón de pausa, todos los procesos reanudarán lo que estaban haciendo, sin reiniciarse ni perderse.

6. Archivos

Para el correcto funcionamiento 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 al efecto de sonido de disparo y la risa robótica (explicada en *Bonus*) incluidos en la carpeta `sonido`.

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

- `puntajes.txt`: Este archivo almacenará todos los resultados de cada partida, guardando el nombre del usuario y el puntaje máximo que ha obtenido en el juego.
- `parametros.py`: En esta tarea, deberás crear este archivo y agregarle todos los parámetros señalados en el enunciado, además de cualquier otro que veas pertinente añadir, como los *paths* y rutas de archivos.

6.1. *Sprites*

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

- **`Aliens`**: esta carpeta consta con tres tipos de aliens y 2 estados para cada uno. Los nombres de cada uno son del estilo `{AlienX.png}` , mientras que el estado *dead* de cada *alien* tendrá un formato `{AlienX_dead.png}`.
- **`Terminator-Dog`**: se cuenta con varios estados del Terminator-Dog. Primero, se tiene dos estados del perro individualmente, con formato `{DogX.png}`. Por otro lado, se cuenta con 3 `sprites` del perro con cada *alien* en estado *dead* mencionados anteriormente. Adicionalmente, se inclue una imagen llamada `{Sprites_Terminator-Dog.png}` que contiene diferentes posiciones del perro, en caso que desees decorar tu tarea con ellos.
- **`Fondos`**: contiene los 3 fondos posibles que determinan el nivel del juego.
- **`Elementos juego`**: contiene una imagen de la bala que utilizará en las estadísticas del juego. Además, almacena los dos tipos de disparadores, `{Disparador_negro.png}` corresponde al disparador que se utiliza para el movimiento, mientras que `{Disparador_rojo.png}` se utiliza para realizar el disparo. Se incluye el `sprite` `{Bala.png}` y las tres fases diferentes de la explosión que se produce al disparar a un *alien*.
- **`Bonus`**: esta carpeta contiene tres `sprites` utilizados en los bonus (ver la sección de *Bonus*). Estos son: `{Bala_extra.png}`, `{Bomba_hielo.png}`, `{Estrella_muerte.png}` .

6.2. Sonidos

Esta carpeta contendrá los dos sonidos que debe implementarse en el juego, el cual se trata de `disparo.wav` y `risa_robotica.wav`. Esta última se implementa dentro de las condiciones específicadas en la sección de *Bonus*.

6.3. `puntajes.txt`

Este archivo se encargará de mantener un registro de **todos los jugadores** del juego *DCComando espacial*. Cuando se finaliza una partida, se debe guardar el puntaje final del jugador en el archivo `puntaje.txt`, en el formato **usuario, puntaje**. Esta nueva fila debe posicionarse debajo de las ya existentes. Posteriormente, esta información debe ser mostrada en la **ventana de ranking**, señalando los 5 mejores puntajes de todo el archivo. Cabe recalcar que este archivo se entregará con datos existentes de registros de jugadores, por lo que debes considerar estos datos al buscar los mayores puntajes. Un ejemplo de cómo debería verse es presentado a continuación:

```
Matiasmasjuan,890
Gatochico,760
Maxy15,430
Diegocostares,370
Emiliax16,210
```

6.4. `parametros.py`

En este archivo se encontrarán 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.

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 sestá 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.

7. *Bonus*

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

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

[4] Para mas información revisar el material de modularización de la semana 0.
[5] Esta nota es sin considerar posibles descuentos.

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. Bonus Risa Terminator-Dog (2 décimas)

Para darle más protagonismo a *Terminator-Dog*, puedes implementarle una risa característica al personaje. Esta risa se encuentra dentro de la carpeta de Sonidos y se utiliza de la siguiente manera:

- `risa_robotica.wav`: este sonido debe reproducirse una vez siempre y cuando *Terminator-Dog* esté en pantalla con un *alien* en la mano.

7.2. Bonus Estrella de la muerte (2 décimas)

Para aumentarle la dificultad a tu partida, puedes incluir la famosa **estrella de la muerte**, la cual puede aparecer aleatoriamente en cualquier parte de la partida (lo que considera cualquier nivel y ambiente de la ventana) y debe durar una cantidad de `TIEMPO_ESTRELLA` en pantalla. La función principal de esta figura será evitar a toda costa que le dispares, ya que son la unidad principal que monitorea el comportamiento de los *aliens*. En caso de que le dispares, perderás un valioso `TIEMPO_PERDIDO` en tu partida. **Esta pérdida de tiempo se efectúa únicamente en esa partida y nivel específico**.

Figura 14: *Sprite* de la estrella de la muerte

7.3. Bonus Bomba de hielo (2 décimas)

Al igual que la estrella de la muerte, también se puede implementar la aparición aleatoria de una**bomba de hielo**, la cual no tendrá movimiento y durará en la ventana un total de `TIEMPO_BOMBA`. En caso de dispararle, todos los *aliens* presentes en la pantalla deben **congelarse** y detener totalmente su movimiento por un `TIEMPO_CONGELAMIENTO`.

Figura 15: *Sprite* de bomba de hielo

7.4. Bonus Disparos extra (2 décimas)

Si quieres disparos extras, puedes implementar una **bala dorada** que aparecerá en algún lugar de la pantalla por un tiempo de `TIEMPO_BALA`. Al seleccionar esta bala (mediante un *click*), se suman 3 balas al contador del panel de estadística. Esta bala sólo aparecerá una vez por partida, por lo que luego de seleccionarla debe desaparecer.

Figura 16: *Sprite* de bala extra

8. Avance de tarea

En esta tarea, el avance corresponderá a manejar apropiadamente **dos tipos diferentes de colisiones** y realizar el **movimiento** del arma.

Para ello, debes crear una ventana en donde se encuentre la **mira** y un **alien**. El arma debe poder moverse según lo indicado en el enunciado, es decir, mediante las flechas `WASD` a una velocidad determinada. Deberás verificar que la mira no se salga de los límites de la ventana.

Además, deberás implementar el movimiento del **alien** a través de una velocidad determinada. En caso de que el alien choque con uno de los bordes de la ventana, se deberá cambiar la velocidad acorde al efecto de rebote que se especifica en el enunciado. No es necesario implementar la animación de disparo.

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.pdf
- Carpeta de *Sprites* entregada junto al enunciado.
- Carpeta de *Sonidos* entregada junto al enunciado.
- Archivo `puntajes.txt`.

Recuerda **no ignorar tus archivos 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.

10. Entregas atrasadas

Posterior a la fecha de entrega de la tarea se abrirá un formulario de Google Form, en caso de que desees que se corrija un *commit* posterior al recolectado, deberás señalar el nuevo *commit* en el *form*.

El plazo para rellenar el *form* será de 24 horas, en caso de que no lo contestes en dicho plazo, se procederá a corregir el *commit* recolectado.

11. Importante: Corrección de la tarea

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

Cuando se publique la distribución de puntajes, se señalará con color:

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

En tu archivo `README.md` deberás señalar el archivo y la línea donde se encuentran definidas las funciones o clases relacionados a esos ítems.

Finalmente, si durante la realización de tu tarea se te presenta algún problema o situación que pueda afectar tu rendimiento, no dudes en contactar al ayudante jefe de Bienestar al siguiente correo: **bienestar.iic2233@ing.puc.cl**.

12. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.8.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del **foro** si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. **Tendrás hasta 2 horas después del plazo de entrega** de la tarea para subir el `README` a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la **guía de descuentos**.
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

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