



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE  
ESCUELA DE INGENIERÍA  
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 Programación Avanzada (I/2018)

## Tarea 5

### Entrega

- Entregable obligatorio
  - **Fecha/hora:** domingo 3 de junio del 2018, 23:59 horas.
  - **Lugar:** <https://goo.gl/forms/DNKA5HDGjiP6s3Do2>
- Tarea obligatoria
  - **Fecha/hora:** domingo 10 de junio del 2018, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: **Tareas/T05/**
- README.md obligatorio
  - **Fecha/hora:** lunes 11 de junio del 2018, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: **Tareas/T05/**

### Objetivos

- Aplicar conocimientos de creación de interfaces con PyQt 5
  - 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

<b>1. Introducción</b>	<b>3</b>
<b>2. <i>Code with Fire</i></b>	<b>3</b>
2.1. Personaje principal [12 %]	3
2.1.1. Movimiento	3
2.1.2. Vida	3
2.1.3. Bombas	4
2.2. Enemigos [11 %]	4
2.2.1. Tipos de enemigos	4
2.2.2. Rango de visión	5
2.3. Colisiones [11 %]	5
2.4. Mapa [5 %]	5
2.5. Elementos del juego [21 %]	6
2.5.1. Inicio de la partida	6
2.5.2. Aparición de enemigos	6
2.5.3. Cambio de <i>stats</i> de enemigos	6
2.5.4. <i>Power-ups</i>	6
2.6. Puntaje [3 %]	7
2.7. Interfaz del juego [17 %]	7
2.7.1. Teclado	8
2.8. Representación gráfica	8
<b>3. Parámetros [2 %]</b>	<b>8</b>
<b>4. <i>Bonus</i></b>	<b>9</b>
4.1. <i>Kick the bomb</i> [5 décimas]	9
4.2. Modo multijugador [5 décimas]	10
<b>5. Consideraciones [16 %]</b>	<b>10</b>
<b>6. Entregable [2 %]</b>	<b>10</b>
<b>7. Restricciones y alcances</b>	<b>10</b>

## 1. Introducción

¡Sorpresa! Programar el parque de diversiones fue *pan comido*. Anonadado por tus habilidades, Javier Pentos decidió contratarte como único programador en su innovadora idea de juego: *Code with Fire*. Este único y gran videojuego consiste básicamente en destruir a tus enemigos y algunos muros usando bombas<sup>1</sup>.

## 2. *Code with Fire*

*Code with Fire* es un juego de un único jugador. En este juego, tú controlas un personaje principal, quien tiene la habilidad de poner bombas. Estos artefactos pueden destruir a los enemigos, algunos muros y **a ti mismo**. El juego ocurre en un mapa, el cual es una grilla bidimensional que contendrá muros tanto destructibles como indestructibles. Al romper ciertos muros pueden aparecer *power-ups* capaces de mejorar los parámetros de tu personaje. Para adquirir estas mejoras, tendrás que pasar sobre ellas.

El juego consiste en lograr que el personaje sobreviva en un mapa acumulando la mayor cantidad de puntaje posible. Durante la partida habrá enemigos que aparecerán con cierta frecuencia. El juego termina cuando el personaje principal pierde **todas** sus vidas.

### 2.1. Personaje principal [12 %]

#### 2.1.1. Movimiento

El movimiento, tanto del personaje principal como de los enemigos, será continuo. Esto significa que el movimiento se efectúa por píxeles. El personaje se mueve en ángulos fijos de 90 grados ( $\rightarrow$ ,  $\uparrow$ ,  $\leftarrow$ ,  $\downarrow$ ) respetando las murallas. Tu personaje debe avanzar una cantidad fija de píxeles —determinada por el programador— mientras se mantienen presionadas las teclas de movimiento. Al iniciar el juego, tanto los enemigos como el personaje principal tienen igual velocidad de movimiento igual a `VEL_MOVIMIENTO`.

#### 2.1.2. Vida

Tanto el personaje principal como los enemigos poseen un atributo de vida. El personaje principal poseerá inicialmente `CANT_VIDAS` máximas, cuyo valor por defecto será 3. Los enemigos sólo poseen una vida.



El personaje perderá una vida en caso de entrar en contacto con un enemigo o con una explosión. Si el personaje se queda sin vidas, el juego terminará.

---

<sup>1</sup>Cualquier parecido con *Bomberman* es mera coincidencia. Pueden encontrar una idea de este juego en el siguiente enlace: [http://www.retrogames.cz/play\\_085-NES.php?emulator=js](http://www.retrogames.cz/play_085-NES.php?emulator=js)

### 2.1.3. Bombas

Las bombas son el método de ataque del juego. Estas restan una vida tanto al personaje principal como a los enemigos que sean alcanzados por su explosión, y también son capaces de destruir muros destructibles. Al apretar la barra de espacio se debe poder colocar una bomba en el centro de la celda en que se encuentra el centro del personaje. Sólo se puede poner bombas en espacios vacíos. Una vez que una bomba es puesta en el mapa, ésta explota después de `TIEMPO_EXPLOSION` segundos, y luego desaparece. El alcance de la explosión se extiende siempre en forma de cruz, en las cuatro direcciones de movimiento del juego hasta un rango máximo, cuyo valor queda a criterio del desarrollador, o hasta topar con algún muro, sea destructible o no.

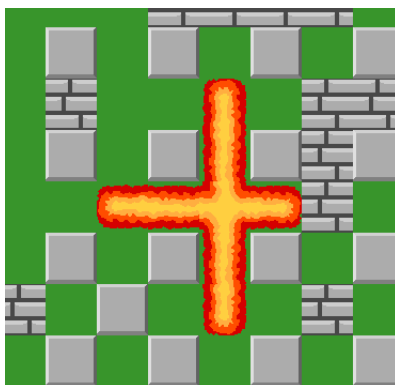


Figura 1: Ejemplo de forma y alcance de una explosión con rango 2.

Cada vez que ocurre una explosión, si el personaje principal o algún enemigo está dentro del rango de la explosión, este perderá una vida. Si el personaje principal fue alcanzado, después de perder la vida será inmune a cualquier otra fuente de daño durante `TIEMPO_INMUNE` segundos, a menos que haya perdido todas sus vidas. Para la tarea se pide que en el instante que una bomba detona, la explosión tome el rango y alcance máximo correspondiente, y que se represente en la interfaz tal como se muestra en la figura 1. Sólo puede haber una bomba en el mapa a la vez, sin embargo esto puede modificarse a medida que avanza el juego mediante el uso de *power-ups*<sup>2</sup>.

## 2.2. Enemigos [11 %]

El personaje principal pierde una vida cada vez que toca a un enemigo. Después de eso, el personaje principal será inmune a cualquier otra fuente de daño durante `TIEMPO_INMUNE` segundos, a menos que haya perdido su última vida. Como cada enemigo tiene sólo una vida, al interactuar con una bomba, éstos mueren y desaparecen.

### 2.2.1. Tipos de enemigos

Existen dos tipos de enemigos que deben poder ser diferenciados en cuanto a la imagen que lo representa:

- **No hostil:** estos enemigos se mueven de manera aleatoria. Cada dos segundos eligen una dirección dentro del mapa, donde todas las direcciones tienen la misma probabilidad. Cuando chocan con un objeto, mantienen su dirección.
- **Hostil:** estos enemigos se mueven inicialmente de manera aleatoria, pero cuando el personaje principal entra en su rango de visión, comienzan a seguirlo y no se detienen (hasta su muerte). El seguimiento de los enemigos será por medio del camino más corto, definido como el camino que reduzca la distancia

---

<sup>2</sup>Ver sección 2.5.4

euclidiana entre el enemigo y el personaje. Si el enemigo se encuentra con un obstáculo en el eje  $X$ , entonces seguirá moviéndose en el eje  $Y$  hacia la posición del personaje, y viceversa. El enemigo también puede quedarse detenido en ambos ejes (*i.e.* puede quedar atrapado).

### 2.2.2. Rango de visión

Los enemigos hostiles poseen un rango de visión. El rango de visión se define de forma circular, cuyo centro es el enemigo y el radio es `RANGO_VISION`.

## 2.3. Colisiones [11 %]

El juego debe ser capaz de detectar colisiones entre las siguientes entidades:

- Personaje principal y bordes del mapa
- Personaje principal y enemigos
- Personaje principal y bombas
- Personaje principal y *power-ups*
- Personaje principal y muros indestructibles
- Personaje principal y muros destructibles
- Enemigos y bombas
- Enemigos y muros indestructibles
- Enemigos y muros destructibles

La distancia entre entidades para detectar colisión queda a criterio del desarrollador, mientras se pueda entender gráficamente.

## 2.4. Mapa [5 %]

El juego transcurre en un mapa que viene dado por el archivo `mapa.txt`. En este mapa se definen **tres** tipos de elementos —representados por 0 (es un **cero**), X y P— todos los cuales ocupan la misma cantidad de espacio dentro del mapa. Dentro del archivo cada elemento viene separado por espacios, y cada fila de la grilla por saltos de línea. La simbología está dada por el cuadro 1. Cada elemento del mapa ocupa un espacio cuadrado de  $N \times N$  píxeles, donde  $N$  queda a criterio del desarrollador. Como el movimiento del personaje es continuo (se avanza por píxeles), si hay dos casillas vacías seguidas el personaje puede tener una parte de su cuerpo en una casilla, y otra parte en la adyacente. Para efectos de esta tarea, se puede asumir que nunca se podrá estar entre más de dos celdas: se utilizarán archivos (`mapa.txt`) que cumplan con esto.

- **Espacio vacío:** Son los únicos lugares donde el personaje y los enemigos pueden desplazarse, y también el único lugar donde puede haber bombas.
- **Muro indestructible:** Son muros que estarán fijos durante toda la partida. Bloquean el paso del personaje y los enemigos. No pueden moverse ni ser destruidos, y limitan el alcance de una explosión.
- **Muro destructible:** Son muros que bloquean el paso del personaje y los enemigos, pero pueden ser destruidos por bombas. Al recibir el alcance de una explosión, el muro limita el alcance de la explosión, pero además es destruido. Cuando un muro destructible desaparece, es reemplazado por un espacio vacío o, posiblemente, un *power-up*.

Símbolo	Significado
0	Espacio vacío
X	Muro indestructible
P	Muro destructible

Cuadro 1: Simbología

X	X	X	X	X	X	X	X	X
X	0	0	0	0	0	0	0	X
X	0	X	P	X	0	X	0	X
X	0	P	P	P	0	0	0	X
X	0	X	P	X	0	X	0	X
X	0	0	0	0	0	0	0	X
X	0	X	0	X	P	X	0	X
X	0	0	P	P	P	0	0	X
X	X	X	X	X	X	X	X	X

Cuadro 2: Ejemplo de `mapa.txt`

## 2.5. Elementos del juego [21 %]

### 2.5.1. Inicio de la partida

Antes de iniciar la partida, se debe desplegar el mapa, con todos los elementos indicados en la sección 2.4. En la ventana del juego debe existir un espacio designado fuera del mapa, en donde se encuentre el personaje. Para iniciar la partida, el jugador deberá arrastrar al personaje con el *mouse* hasta un lugar permitido en el mapa, es decir, un espacio vacío. Una vez que el personaje se encuentre en el mapa, el juego comenzará. A partir de ese momento, los enemigos aparecerán según lo indicado en la sección 2.5.2.

### 2.5.2. Aparición de enemigos

Cuando se inicia la partida habrán tres enemigos: dos no hostiles y uno hostil. Luego cada cierto tiempo aparecerán enemigos de ambos tipos. Un enemigo siempre debe aparecer en lugares donde hay un espacio vacío y a una distancia mínima del personaje que queda a criterio del desarrollador. El tiempo de aparición para cada enemigo será el siguiente:

- **Enemigos no hostiles:** aparecerán cada cierto tiempo, con una distribución uniforme con parámetros (A.NO\_HOSTIL, B.NO\_HOSTIL).
- **Enemigos hostiles:** aparecerán cada cierto tiempo, con una distribución exponencial con parámetro (LAMBDA\_HOSTIL).

### 2.5.3. Cambio de *stats* de enemigos




A medida que el jugador incrementa su puntaje, los enemigos sufrirán modificaciones, que les permitirán mejorar sus atributos (*stats*). El cambio de *stats* se realizará cada vez que el jugador obtenga AUMENTO\_DIFICULTAD puntos. Los cambios serán los siguientes:

- **Aumento de velocidad:** los enemigos aumentarán su velocidad en 50 %.
- **Aumento de enemigos:** se disminuyen a la mitad los parámetros de distribución del tiempo de aparición que tienen los enemigos.



### 2.5.4. *Power-ups*

El juego contendrá distintos tipos de *power-ups* que afectarán al personaje principal, quien los adquiere al colisionar con ellos. Los muros destructibles tienen probabilidad de 30 % de dejarle *power-ups* al jugador. Todos los *power-ups* tienen la misma probabilidad de salir. Existen categorías de *power-ups*.

- **Mejoras:** duran por el resto del juego y tienen efecto en alguno de los atributos del personaje.

-  **Vida:** recupera una vida. No se puede exceder el máximo de vidas.
-  **Cantidad de bombas:** aumenta en 1 la cantidad máxima de bombas que puede haber en el mapa al mismo tiempo.
-  **Velocidad:** multiplica la velocidad del personaje por 1,25.

- **Poderes:** mejoran de manera extrema los atributos del personaje, pero sólo duran un corto tiempo.

-  **Supervelocidad:** aumenta la velocidad de movimiento del jugador al triple durante 10 segundos.
-  **Juggernaut:** el jugador se vuelve invencible. Las bombas y los enemigos no le hacen daño durante 5 segundos.

## 2.6. Puntaje [3 %]

A lo largo de una partida, el personaje ganará puntaje de acuerdo a su desempeño.

- Por cada enemigo eliminado, el jugador será bonificado con `PUNTAJE_ENEMIGO` puntos.
- Por cada `TIEMPO` segundos que el jugador sobreviva, recibirá una bonificación de `PUNTAJE_TIEMPO` puntos.
- Por cada muro destructible que el jugador destruya con una de sus bombas, será bonificado con `PUNTAJE_PARED` puntos.

A medida que el jugador tenga mayor puntaje, el juego aumentará su dificultad, es decir, aparecerán enemigos con mayor frecuencia (2.5.2) y tendrán mejores *stats* (2.5.3).

## 2.7. Interfaz del juego [17 %]

Al iniciar el programa, se debe mostrar una ventana con un menú principal. En este menú debe tener dos opciones: (1) iniciar el juego (*start*) y (2) mostrar *ranking* de los mejores puntajes (*highscores*).

Al presionar la opción *highscores*, el usuario debe ser capaz de observar los 10 mejores puntajes obtenidos hasta ese momento, junto a los nombres de los jugadores que lo obtuvieron. Estos puntajes tienen que perdurar entre ejecuciones del juego.

Al presionar la opción *start*, se debe pedir un nombre de jugador, que por defecto será **Jugador1**. A continuación, el usuario ingresará a la partida en una nueva ventana. Esta nueva ventana debe cumplir con los siguientes requisitos.

- Botón de **Salida:** debe ser capaz de terminar la partida actual, registrando los puntajes obtenidos hasta el momento y regresando al jugador al menú inicial. Esta opción también debe activarse al presionar `Ctrl+E`.
- Botón de **Pausa:** Debe ser capaz de poner el juego en pausa. Esta opción también debe activarse al presionar `Ctrl+P`.

- **Puntaje:** se debe visualizar el puntaje actual del jugador, y ser actualizado a medida que cambie. Se debe incluir un *progress bar* que indicará el puntaje actual con respecto al puntaje necesario para el aumento de dificultad. Inicialmente estará en 0, y el máximo será `AUMENTO_DIFICULTAD`. Este indicador se actualizará cada vez que el jugador obtenga puntos y, una vez que alcance el límite, volverá a 0.
- **Vida:** se debe indicar la cantidad de vidas actuales del jugador, y se debe actualizar cada vez que cambia.
- **Poderes:** debe ser posible visualizar todos los *power-ups* que adquiera el jugador durante la partida. En caso de que se trate de un *poder*, deberá visualizarse sólo durante su duración.
- **Personaje:** debe existir un espacio en donde se encuentra el personaje antes de iniciar la partida. El jugador debe ser capaz de arrastrar al personaje con el *mouse* a un lugar permitido en el mapa (2.4).

Una vez finalizada la partida, debe aparecer la información con los *highscores*, incluyendo el puntaje recientemente obtenido, en caso de estar en los Top 10.

### 2.7.1. Teclado

- ← : Moverse hacia la izquierda.
- → : Moverse hacia la derecha.
- ↑ : Moverse hacia arriba.
- ↓ : Moverse hacia abajo.
- Spacebar : Poner bomba.
- Ctrl+P: Pausa.
- Ctrl+E: Salida.

## 2.8. Representación gráfica

Los personajes deben poseer movimientos animados, tanto para avanzar como para atacar. Para lograr esto, se debe diferenciar su aspecto al moverse, considerando al menos tres estados de movimiento para cada acción y dirección, como los mostrados a continuación:



De esta forma, el movimiento se verá más fluido. Para esta sección, se recomienda buscar *sprites* en internet. Junto al enunciado se les entregará un conjunto básico de *sprites* para el entorno de su interfaz.

## 3. Parámetros [2 %]

Para esta tarea, todos los parámetros (constantes) deben ser definidos en un archivo (de extensión `.py`) separado, que tenga por nombre `parameters.py`. De esta manera, podrán ser modificados con facilidad, permitiendo evaluar su juego bajo distintos valores iniciales.

Entre los principales parámetros a considerar se encuentran:



- VEL\_MOVIMIENTO: velocidad de movimiento del personaje principal.
- CANT VIDAS: cantidad de vidas del personaje principal.
- TIEMPO\_EXPLOSION: tiempo en segundos que demora una bomba en explotar.
- TIEMPO: tiempo en segundos que debe transcurrir para que el jugador reciba puntos por tiempo.
- PUNTAJE\_TIEMPO: cantidad de puntos otorgados por sobrevivir cierta cantidad de tiempo.
- PUNTAJE\_ENEMIGO: cantidad de puntos recibidos por enemigo eliminado.
- PUNTAJE\_MURO: puntaje otorgado por destruir un muro.
- AUMENTO\_DIFICULTAD: puntaje necesario para que ocurra un aumento en la dificultad.
- RANGO\_VISION: rango de visión que tendrán los enemigos.
- TIEMPO\_INMUNE: tiempo en el que estarás inmune al rango de visión y daño luego de ser alcanzado por un enemigo
- A\_NO\_HOSTIL, B\_NO\_HOSTIL: parámetros de la distribución del tiempo de aparición de enemigos no hostiles.
- LAMBDA\_HOSTIL: parámetro de la distribución del tiempo de aparición de enemigos hostiles.
- N: tamaño (en pixeles) de la grilla que corresponde al mapa.

Cualquier otro parámetro que consideres necesario debe estar incluido en este archivo.

## 4. *Bonus*

Para hacer más entretenida y amigable nuestra tarea, hemos decidido proponerles dos *bonus*.

### 4.1. *Kick the bomb* [5 décimas]

No nos quedamos satisfechos con nuestras bombas estáticas y aburridas. Es por esto que hemos decidido darles la opción de implementar bombas móviles, en lugar de las bombas por defecto. Estas nuevas bombas tienen las siguientes características:

- Una vez puesta la bomba, el jugador puede avanzar hacia la dirección de la bomba para empujarla en la dirección y sentido de la colisión con el jugador (es decir, si el jugador choca con la bomba desde abajo, ésta avanzará hacia arriba).
- Las bombas empujadas no se detienen hasta que colisionan con un muro, ya sea destructible o no, o con un enemigo.
- Los centros de las bombas se deben mover por los centros de los casillero por los cuales avanza. El movimiento tiene que ser de manera continua. En esta situación, puede ocurrir que las bombas no queden en el centro de una grilla en la posición del eje por el cual son empujadas. Esto ocurrirá si la bomba choca con un enemigo que no está en un centro exacto de una casilla, ya que esto hará que la bomba se detenga en una posición que no necesariamente será el centro de la casilla adyacente al enemigo con el que choca.
- Debido al punto anterior, la explosión avanzará (si es que puede) en la casilla donde está el centro de la bomba y en las direcciones a las casillas adyacentes a esta casilla.
- La velocidad de la bomba queda a criterio del desarrollador.

## 4.2. Modo multijugador [5 décimas]

Como bien podemos intuir, a *Code with Fire* le hace falta un modo **multijugador** para poder jugar contra algún amigo o familiar. Es por esto que se propone implementar un modo de dos jugadores. Los requisitos son:

- El jugador 2 debe moverse con WASD, donde W, A, S, D son para subir, ir a la izquierda, bajar e ir a la derecha respectivamente, y poner la bomba con la tecla F. El jugador 1 debe utilizar las flechas ( $\uparrow$ ,  $\leftarrow$ ,  $\downarrow$ ,  $\rightarrow$ ) para moverse. La tecla para activar la bomba del primer jugador será la barra espaciadora, es decir, las teclas del jugador 1 se mantienen.
- Los elementos de la interfaz como puntaje, vida y poder, deben visualizarse para cada jugador.
- Se mantienen las propiedades de las bombas. Las bombas pueden dañar a cualquier jugador.
- Una vez que un jugador es eliminado el juego, continúa hasta que el jugador que quedaba pierda su última vida. Este último será el jugador vencedor.
- Si se implementó el *bonus Kick the bomb*, los jugadores podrán empujar las bombas de cualquier jugador. Ahora las bombas se detienen cuando chocan con un muro, otra bomba, un enemigo u otro jugador.

## 5. Consideraciones [16 %]

Para esta tarea se pedirá como requisito la utilización de señales para la interacción entre las distintas entidades de la tarea.

## 6. Entregable [2 %]

Se debe entregar un diagrama de clases con todo el modelamiento del problema. Esto incluye las clases junto con sus atributos, métodos y *properties*, y todas las relaciones existentes entre éstas (asociación, composición y herencia), con el objetivo de distinguir y separar el *back-end* y el *front-end*.

## 7. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el *Código de honor de la Escuela*: haz clic aquí para leer.
- Tu programa debe ser desarrollado en Python v3.6.
- Tu código debe seguir la guía de estilos descrita en el PEP8.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibida. Pregunta en el foro si es que es posible utilizar alguna librería en particular.
- El ayudante puede castigar el puntaje<sup>3</sup> de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debes adjuntar un archivo `README.md` donde comentes sus alcances y el funcionamiento del sistema (*i.e.* manual de usuario) de forma *concisa y clara*. **Tendrás hasta 24 horas después de la fecha de entrega** de la tarea para subir el `README.md` a tu repositorio.

---

<sup>3</sup>Hasta 5 décimas.

- Crea un módulo para cada conjunto de clases. Divídelas por las relaciones y los tipos que poseen en común. **Se descontará hasta un punto si se entrega la tarea en un solo módulo**<sup>4</sup>.
- 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).

---

<sup>4</sup>No agarres tu código de un solo módulo para dividirlo en dos; separa su código de forma lógica