

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores (CE)

Introducción a la Programación (CE-1101)

Profesor Jeff Schmidt Peralta

## Proyecto 1

# Vintage Bomberman

Rodrigo Arce Bastos

Carné: 2024115582

I Semestre 2024

# Introducción

Todos los objetivos del proyecto se han cumplido. Primero que todo, se ha elaborado el juego utilizando enteramente recursividad, así como la utilización de Tkinter para el desarrollo de la interfaz gráfica. Tkinter es una herramienta que, aunque no esté diseñada específicamente para el desarrollo de juegos, proporciona algunos elementos bastante útiles. Se desarrolló un video del timelapse de la creación del juego. [Proyecto de Introducción a la programación: bomberman \(youtube.com\)](https://www.youtube.com/watch?v=bomberman).

## Mecánicas y reglas del juego

Voy a explicar las reglas del juego.

Figura 1. Pantalla de juego



En la figura 1, se puede observar la pantalla de juego principal. Esta se conforma de 2 canvas: El canvas de información y el canvas del juego. En el canvas de información, se encuentra, de izquierda a derecha: El tiempo, nombre de jugador, las vidas, el puntaje actual, puntaje mínimo, bombas restantes y el cuadrito que muestra si se ha obtenido la llave. El jugador es colocado, inicialmente, en la esquina superior izquierda, y tiene una cantidad de bombas equivalente a la cantidad de bloques destructibles. Esto porque estamos tomando el peor de los casos (worst-case-scenario), ya que no se sabe exactamente dónde está ni la llave ni la puerta; además, se toma en cuenta que se tiene que matar, como mínimo, 2, 3 y 4 enemigos (en promedio) en cada nivel respectivamente. Todos los niveles son posibles. El jugador, **se mueve con las flechas del teclado y pone bombas con la barra espaciadora.**

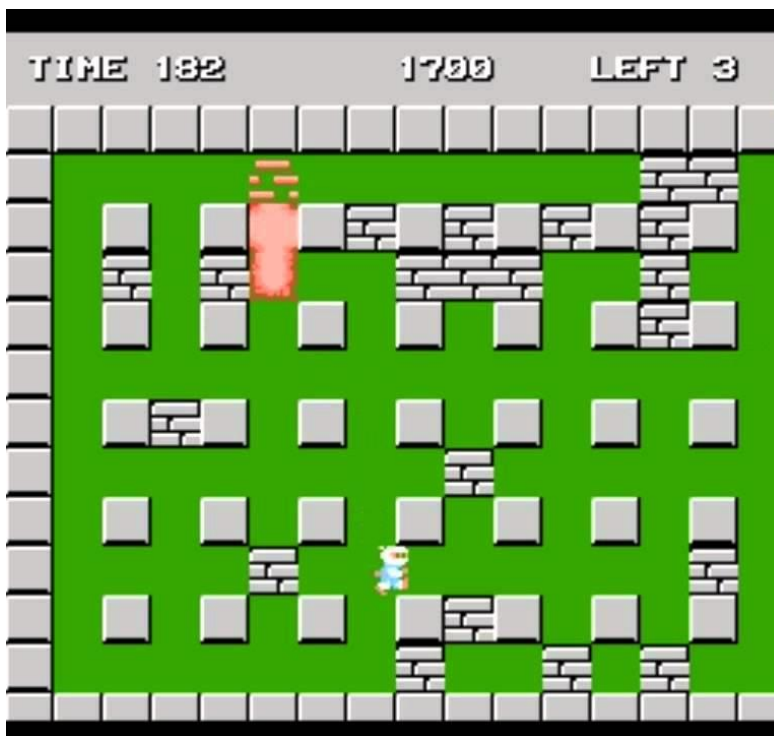
## Descripción del problema

En este caso, se quiere desarrollar nuestra propia versión de Bomberman utilizando recursividad. Esto es un gran reto, pues al no poder utilizarse POO, se tiene que recurrir a las variables globales, las cuales pueden resultar más confusas. También, esto implica la necesidad de una función para reiniciar todas estas variables globales iniciales. La clave para desarrollar bien este proyecto, era crear soluciones **eficientes** ante los problemas, pues hay varias cosas que están actualizándose al mismo tiempo en pantalla: movimiento de jugador, la explosión de la bomba, el movimiento de cada uno de los enemigos, la actualización en tiempo real de las estadísticas del jugador, entre otros.

### Algoritmos utilizados

A continuación, voy a explicar cómo funciona el código de cada nivel. Primero que todo, se genera la base del nivel. Si nos fijamos en el Bomberman clásico, el nivel está compuesto por las estadísticas del jugador y por el nivel en sí. Todo nivel de Bomberman, tiene la misma base: una pared indestructible, compuesta de varios bloques del mismo tipo, y dentro tiene otros bloques del mismo tipo, bloque por medio, así como se ve en la figura 1.

Figura 2. Bomberman de NES



Entonces, desarrollé un algoritmo que creara la base del nivel. Ahora, ¿Qué diferencia a cada nivel? Los bloques destructibles, su frecuencia de generación y la cantidad de enemigos. Entonces, para esto, no ocupamos “hardcodear” cada uno de los niveles. Podemos,

simplemente, generar los bloques destructibles mediante el uso de probabilidades, las cuales van a ir aumentando conforme se pasan los niveles. Entonces, primero creamos una matriz bidimensional, la cual va a agregar las coordenadas de los bloques destructibles. Después de esto, creamos el algoritmo que se va a encargar de seleccionar espacios vacíos aleatorios dentro del canvas, para agregar las coordenadas x, y de nuestro bloque destructible. Que conste que, en esta parte del algoritmo, aún no estamos agregando los bloques al canvas en sí, porque aún falta seleccionar dos bloques destructibles aleatorios: uno para la llave y otro para la puerta. ¿Por qué no se pueden generar de la misma manera y ya? En el caso de los bloques destructibles, no nos importa realmente si se generan muchos o se generan pocos, así que simplemente podemos utilizar una probabilidad. Sin embargo, si hacemos esto con la llave, por ejemplo, hay problemas.

1. Si ponemos una probabilidad muy baja, el nivel sería muy fácil y no tendría chiste
2. Si ponemos una probabilidad medianamente alta, como 1/20, hay posibilidades de que no se genere la llave o la puerta, así que es muy arriesgado

Figura 3. Probabilidad de generar un bloque destructible

```
if is_empty_space(x, y) and random.randint(a: 0, 7 - level) == 1:
    breakable.append([x, y])
```

Por esto, una vez generada toda la lista de bloques destructibles, agarramos dos coordenadas aleatorias: una para la llave y otra para la puerta. Una vez se tienen las coordenadas, podemos usar un truco con el canvas. Para esto, generamos primero la llave o la puerta, y luego generamos el bloque destructible. Esto hace que el bloque se superponga, y cuando este sea destruido, sea visible la llave o la puerta.

Figura 4. Algoritmo de generación de la llave y la puerta

```
x, y = b[0]
if [x, y] == trapdoor_coords:
    image_door = ImageTk.PhotoImage(Image.open("api/assets/images/door.png").resize((60, 60)))
    canvas.img_d = image_door
    img = canvas.create_image(
        x, y,
        image=image_door,
        anchor="nw"
    )
    trapdoor_coords = [x, y, x + 60, y + 60, img]

elif [x, y] == key_coords:
    image_key = ImageTk.PhotoImage(Image.open("api/assets/images/key.png").resize((60, 60)))
    canvas.img_k = image_key
    img = canvas.create_image(
        x, y,
        image=image_key,
        anchor="nw"
    )
    key_coords = [x, y, x + 60, y + 60, img]
img = canvas.create_image(
    x, y,
    image=image,
    anchor="nw"
)
collisions.append(
    [x, y, x + 60, y + 60, True, img]
)
```

Sobre la generación de los enemigos, estos se generan después de crear todo el mapa. Para generar los enemigos, obtenemos los paths (caminos) más largos del nivel.

Figura 5. Representación visual de los paths o caminos

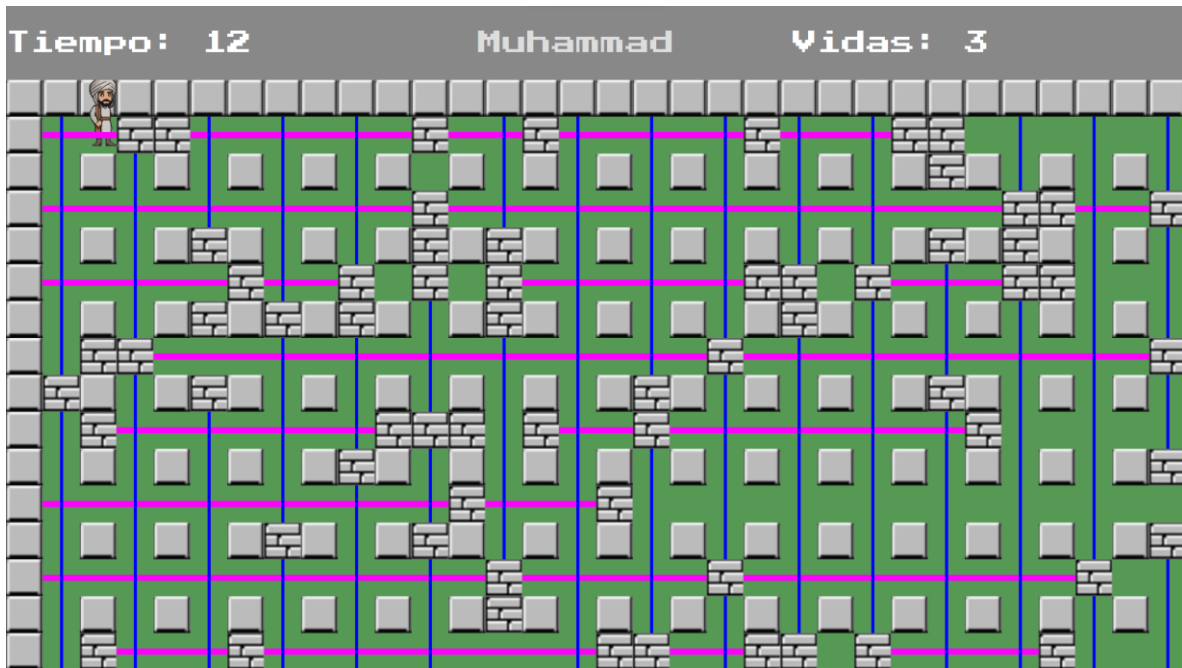


Figura 6. Algoritmo que obtiene estos paths

```
# Agarra los paths más largos
for_replacement(
    lambda x: get_paths_x(canva_level, y=x), j=game_height
)

for_replacement(
    lambda x: get_paths_y(canva_level, y=x, gh=game_height)
)
```

Después, obtenemos los cinco paths más largos y hacemos que los enemigos se muevan infinitamente en este. Esto es más sencillo que hacer que los enemigos detecten una colisión.

### Sistema de puntajes y pantalla de victoria o derrota

Sobre el sistema de puntajes, el jugador gana puntos conforme asesina a personas. El puntaje está definido por la siguiente fórmula.

$$\begin{aligned} s_p &= s_{p-1} + j + v_e^2 \\ s_1 &= 0 \end{aligned} \quad (1)$$

Donde  $s_p$  es el puntaje del jugador en el instante  $p$ ,  $s_{p-1}$  el puntaje anterior,  $j$  es la id del enemigo dentro del canvas, y  $v_e$  es la velocidad del enemigo. El enemigo se mueve 1 pixel por cada 5 milisegundos de juego.

Cuadro 1. Velocidades de los enemigos

Tipo de enemigo	Velocidad (pixeles por cada 5 milisegundos)
Guerrillero	5
Walter White (WW)	7
Agente de CIA	9

La importancia del sistema puntajes radica en que el jugador debe tener un mínimo de puntaje para ganar la partida, el cual es equivalente a

$$S_m = 750 \cdot n \quad (2)$$

Donde  $S_m$  es el puntaje mínimo, y  $n$  es el número del nivel. El puntaje es subido al archivo *settings.json*.

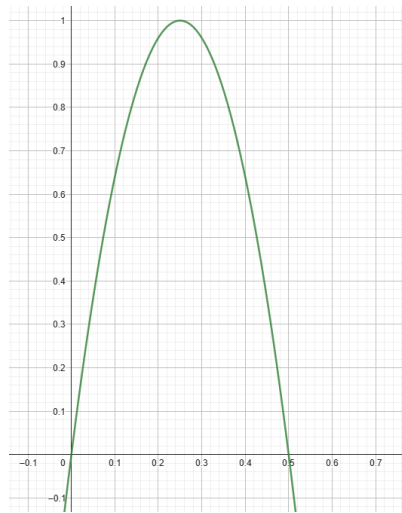
Sobre la pantalla de derrota, esta sucede al morir tres veces. El jugador muere cuando es golpeado por un enemigo, o por la explosión de una misma bomba. Sobre la pantalla de victoria, esta sucede al completar el nivel exitosamente.

### Sobre la bomba

La propagación de la bomba, está definida con la siguiente función cuadrática

$$r(t) = -16\left(x - \frac{1}{4}\right)^2 + 1$$
$$0 \leq t \leq \frac{1}{2} \quad (3)$$

Figura 7. Gráfica de la propagación de la bomba.



Donde  $r(t)$  es un número entre 0 y 1, el cual se utilizará como un número porcentual, para que la bomba tenga una propagación animada y vistosa. La explosión sucede en medio segundo.

Ahora, este número  $r$  lo utilizamos de manera porcentual, aplicada hacia nuestra constante de rango. La cual es  $b_r = 120px$ . Esto es lo que da el efecto de “animación” a la bomba. Ahora, si la explosión detecta una colisión, va a bloquear el ratio para que la explosión no se siga expandiendo en ese eje.

Figura 8. Propagación de la bomba

```
if 0 <= t <= 1 / 2:
    ratio = propagation_ratio(ti, time.time())
    exp_l = bombs[bomb_index][2] = exp_x + bomb_range * (ratio if not locked_ratio_x_p else locked_ratio_x_p)
    # El objetivo de esta secuencia de if's es lockear el ratio a un valor determinado, para que, cuando colisione
    # la bomba no se siga propagando
    if is_collision(exp_l, exp_y, collisions, destroy=True, canvas=canvas, hitrange=0):
        bombs[bomb_index][6] = ratio / 2

    exp_h = bombs[bomb_index][3] = exp_y - bomb_range * (ratio if not locked_ratio_y_p else locked_ratio_y_p)
    if is_collision(exp_x, exp_h, collisions, destroy=True, canvas=canvas, hitrange=0):
        bombs[bomb_index][7] = ratio / 2

    negative_l = bombs[bomb_index][4] = exp_x - bomb_range * (ratio if not locked_ratio_x_n else locked_ratio_x_n)
    if is_collision(negative_l, exp_y, collisions, destroy=True, canvas=canvas, hitrange=0):
        bombs[bomb_index][8] = ratio / 2

    negative_h = bombs[bomb_index][5] = exp_y + bomb_range * (ratio if not locked_ratio_y_n else locked_ratio_y_n)
    if is_collision(exp_x, negative_h, collisions, destroy=True, canvas=canvas, hitrange=0):
        bombs[bomb_index][9] = ratio / 2




    if negative_l <= player_x + 60 <= exp_l and exp_h <= player_y + 60 <= negative_h and not immune:
        t = Thread(target=on_death, args=(canvas,))
        t.start()
```

Cabe destacar que a la función que detecta las colisiones, *is\_collision*, se le agregó un parámetro especial *destroy*, el cual, si es True, destruirá el bloque **destructible** que alcance en ese eje.

Análisis de resultados

A continuación, presentaré un cuadro para cada uno de los escenarios.

Cuadro 2. Resultados

Escenario	Captura	Se logró el objetivo
Bomba		Si
Enemigos		Si
Victoria y muerte		Si



	<div><div>Tiempo: 7    rodri    Vidas: 0   0   1500   14</div><div></div></div>	
Ajustes pantallas adicionales	<div>y</div> <div><div><div><div><div>🚩</div><div><div>LAS FLIPANTES AVENTURAS DEL TÍO BOMBETAS</div></div></div><div><div>Iniciar juego</div><div>Configuración</div><div>Records</div><div>Creditos</div><div>Salir</div></div></div><div><div><div></div><div></div><div></div></div><div><div>Agregar nuevo nombre</div><div>Salir</div></div><div><div>Noticia</div><div>0</div><div>Efectos de sonido</div><div>0</div></div><div><div>Mejores cinco puntajes</div><div>1: Player: 70000</div><div>2: Player: 7000</div><div>3: elpapu: 2832</div><div>4: muhammed: 2670</div><div>5: rodri: 2563</div><div>Salir</div></div><div><div><div>Sobre la universidad</div><div>Sobre el programador</div><div>Sobre el juego</div></div><div><div><div>Estudiante de Ingeniería en Computadores del Instituto Tecnológico de Costa Rica. Este proyecto es parte del curso de Introducción a la Programación (CE-1101), a cargo del profesor Jeff Schaidt Peralta</div><div>Soy Rodri, tengo 19 años, cumplo años el 7 de enero y tengo afición por la informática y la investigación. Mi sueño es trabajar en la investigación de máquinas cuánticas. Me gusta el software libre y el Open Source.</div><div>Bienvenidos a las flipantes aventuras del tío bombetas! El objetivo es asesinar a personas con las explosiones de las bombas. Para ganar, debes alcanzar el mínimo de puntaje (750, 1500 y 2250) para cada nivel, con las limitadas bombas. Entre más asesines a seres humanos, mejor!</div></div><div><div>2024 Costa Rica. B</div></div><div><div><div>Información Personal</div><div><div></div><div><div>Nombre Completo</div><div>Ariel Bastos Rodríguez</div><div>Cédula</div><div>110200718</div><div>Dirección permanente</div><div>Guamacarte, Santa Cruz, Santa Cruz, 200 Metros Sur Del Arroyo Pura, Casa Con Palto Grande Frente Al Nágono "Selección Dorada Rutilo"</div><div>Teléfono</div><div>81200000</div><div>Córeo electrónico</div><div>ariel.110200718@gmail.com</div></div></div><div><div>Información Académica</div><div><div>Carrera</div><div>Ingeniería En Computadores</div><div>Sede</div><div>Campus Tecnológico Central Cartago</div><div>Arredio</div><div>Duma</div><div><div>Clas de Matemás</div><div>Clas de Horario</div></div></div></div><div>Salir</div></div></div></div></div></div></div></div>	Si

Código limpio y documentación	<pre> 1 usage  └ Rodri def generate_borders(level: tk.Canvas, window: tk.Toplevel, size, height):     """     :param level: canvas del nivel     :param window: ventana del nivel     :param size: tamaño de cada bloque     :param height: alto del canvas     :return: None     """      block_image = ImageTk.PhotoImage(         image=Image.open(r"api/assets/images/blocks_1.png").resize((size, size)))     window.block = block_image     recursive_gen_blocks_x(level, block_image, size, start=0)     recursive_gen_blocks_y(level, block_image, size, start=0, height) </pre>	Si
-------------------------------	--	----

## Dificultades encontradas

Lo que tomó más tiempo desarrollar fueron los enemigos, ya que implicó hacer un algoritmo que creara los caminos más largos, y que tomara los primeros 5 más largos. También, las colisiones fueron un problema en un inicio, ya que tenían algunos bugs, sin embargo, se solucionó con una matriz bidimensional para cada bloque que se puede colisionar. Por último, la propagación de la bomba tenía problemas con la hitbox, sin embargo, se solucionó con la siguiente función.

Figura 9. Función que detecta si una entidad está dentro del rango de una explosión

```

def in_explosion(x, y, l):
    """
    Chequea si, en las coords x, y, está explotando
    :param x: eje x
    :param y: eje y
    :param l: bombs
    :return: bool
    """

    if not l:
        return False
    exp_x, exp_y, exp_l, exp_h, negative_l, negative_h, _, __, ____, exploding = l[0]
    if exploding and negative_l and ((negative_l <= x <= exp_l and exp_y - 15 <= y <= exp_y + 15) or (
        exp_x - 15 <= x <= exp_x + 15 and exp_h <= y <= negative_h
    )):
        return True
    return in_explosion(x, y, l[1:])

```

## Bitácora de actividades

Respecto al desarrollo del proyecto, las sesiones, en promedio, duraban en promedio 3 horas. Algunas sesiones llegaban a ser intensivas, y llegaban a 5 y 6 horas. Estas sesiones fueron las del desarrollo de las bombas y el de los enemigos. Respecto al diseño del juego, se utilizó el editor de imágenes *Gimp* y para la edición del video, se utilizó *Davinci Resolve 19*. A continuación, se detalla cada actividad con una tabla y tiempos aproximados.

Cuadro 3. Estadísticas de tiempos

<b>Actividad</b>	<b>Tiempo estimado</b>
Desarrollo del menú principal	<i>40 min</i>
Desarrollo de la pantalla de selección de niveles	<i>20 min</i>
Creación de la base del mapa	<i>1 h</i>
Creación de los bloques destructibles	<i>1 h 30 min</i>
Generación del jugador y movimiento	<i>1 h 20 min</i>
Colisiones	<i>2 h 30 min</i>
Canvas de información	<i>30 min</i>
Creación de la bomba	<i>5 h</i>
Los enemigos	<i>6 h</i>
Búsqueda y arreglo de bugs generales	<i>2 h</i>
Creación de pantallas de configuración y créditos	<i>1 h 30 min</i>
Playtesting	<i>15 min</i>
Creación de documentación	<i>2 h</i>
<b>Total</b>	<b><i>24.6 h</i></b>

## Conclusión

El proyecto me ha enseñado a utilizar correctamente la interfaz gráfica de Tkinter, así como el desarrollo de estas haciendo uso de recursividad. Aunque las limitaciones hayan representado un reto, no vuelven el proyecto imposible de realizar. Hay cosas que noté que podía mejorar pero que, por cuestiones de tiempo, no pude realizar.

1. El algoritmo de los enemigos pudo ser más eficiente, ya que elimino los paths más pequeños. Esto lo podría hacer más eficiente si solamente agarrara los paths más largos y los agregara a una lista.
2. El movimiento pudo haber sido más limpio si se hubieran utilizado binds para eventos de KeyPress y KeyRelease, en vez de hacerlo todo en un solo evento. Al hacer el movimiento en un solo evento, entonces el jugador se mueve “más pegado”

## Referencias

Lista de fuentes utilizadas

1. Píldoras Informáticas. <https://www.youtube.com/@pildorasinformaticas>
2. StackOverflow. <https://stackoverflow.com/>
3. Longplay Bomberman NES. <https://www.youtube.com/watch?v=CZ9Pu9Usk5o>
4. Codemy. <https://www.youtube.com/@Codemycom>
5. Itch.io. <https://itch.io/>