Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias De La Computación

Informe Tarea 2: BomberMan

Integrante: Valentina González Fuentes
19.697.350-8
Sección: 1
Profesores: Nancy Hitschfeld
Auxiliar: Pablo Pizarro
Pablo Polanco
Mauricio Araneda
Fecha entrega: 12 de Junio de 2018

1. Introducción

En este informe se abordará la creación de un juego, en este caso, el juego Bomberman. Este es un juego en el cual un personaje debe atravesar un laberinto, evitando a diversos enemigos.

El objetivo del juego es sobrevivir para encontrar la puerta que permite ganar el juego, la que se encuentran bajo muros destructibles que Bomberman debe destruir con bombas.El personaje está limitado por el tiempo, el cual va disminuyendo hasta llegar a 0. Sí el personaje no logra su objetivo antes que el tiempo se acabe, muere. También hay objetos que pueden ayudar a mejorar las bombas de Bomberman, como poner más bombas a la vez ó el personaje puede tener una vida extra.

En la siguiente figura se encuentra una representación gráfica del juego:

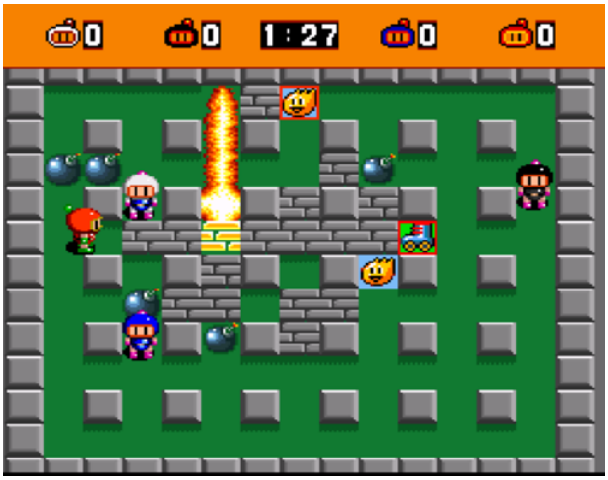

Figura 1.1: Modificación de árbol con respecto a la división

2. Esquema de la solución

2.1. Diseño de Gráficos

Para el diseño de los gráficos del juego, se deben diseñar lo siguientes puntos:

- Personaje principal: Debe ser una figura compuesta por varias primitivas.
- Muros y Muros destructibles.
- Bombas. Al explotar las bombas deben tener un sprite o dibujo que indique la explosión.
- Power Ups. Deben haber 2 diseños de power-ups. En este caso, uno es para aumentar la cantidad de bombas que se pueden colocar y el segundo es una vida extra para el personaje.
- Enemigos: Debe haber 2 diseños para enemigos.

- Escena: Utilizando los diseños anteriores construya una escena estática del juego, donde debe tener por lo menos 4 enemigos.

2.2. Mecánica simple de juego

En esta sección se describen la mécanica del juego y lo que se debe implementar en el juego según el siguiente listado:

- El personaje se puede mover por el laberinto con las teclas arriba, abajo, izquierda y derecha.Además debe poder colocar bombas con la tecla A. El personaje debe chocar contra las murallas y con las bombas.
- Las bombas deben explotar luego de 3 segundos.
- Si el personaje toca a un enemigo o recibe una explosión el juego termina.
- Si un enemigo recibe una explosión es eliminado de la pantalla, así como también sus interacciones.
- Si un muro destructible recibe una explosión debe desaparecer. Debe existir una salida bajo un muro destructible

2.3. Características avanzadas

Son características más díficiles de implementar que le dan más dinamica al juego, las cuales son las siguientes:

- Los muros destructibles aparecen en posiciones aleatorias.
- Los enemigos aparecen en posiciones aleatorias y deben moverse de forma aleatoria.
- Deben implementarse las características de los dos power-ups.

3. Explicación detallada de la solución

La solución se basa a partir del Modelo Vista Controlador, el cual ayuda a organizar las distintas partes para la creación de un juego. Para poder dibujar e interactuar en el juego se crearon clases y objetos, los cuales se detallaran a continuación.

3.1. Personaje

Se crea una clase personaje, pues este tiene atributos propios diferentes a los Enemigos, los cuales son:

- x: Indica la posición x que permite saber donde se encuentra el personaje
- y: Indica la posición y que tiene el personaje.
- vivo: Atributo booleano que permite ver su estatus de vida durante todo el juego.
- amigo: Atributo boolean llamado amigo que verifica que el personaje es amigo, en el caso de los enemigos es False.
- bombas: otorga la cantidad de bombas que posee el personaje, en este caso una bomba a la vez.
- vida: Es de tipo entero y que es igual a 1, ya que el personaje posee sólo una vida, ya que si le explota una bomba al lado de él, el personaje muere.
- rad: Permite más adelante poder detectar las colisiones con otros objetos.
- tipo: Diferencia esta clase con las otras clases, en este caso el tipo es "Personaje".

La clase personaje tiene métodos propios como por ejemplo el dibujar() que se realiza a través de primitivas, mayormente poligonos y circulos como se puede apreciar en la siguiente figura:

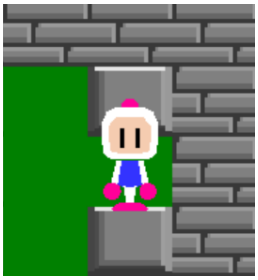

Figura 3.1: Diseño Personaje

Además de este método, posee otro llamado EncuentraSalida(salida), el cual compara si el personaje está en la misma posición que la puerta entregando True y si no False. Esta función sirve para más adelante al construir todo el programa.

3.2. Muro

La clase muro se crea para poder dibujar un tipo de muro para luego implementarlo en toda la escena. Esta clase posee los siguientes atributos :

- x: Posición x.
- y: Posición y.

- tipo: Se crea un atributo tipo que en este caso es "Muro", el cual permite diferenciar esta clase de las otras.
- rad: Atributo radio para ver las colisiones con otros objetos.

La clase muro tiene su propio método dibujar(), el cual se basa a través de primitivas de OpenGl, tal como los cuadriláteros, los cuales poseen varios niveles de iluminación para poder darles más realismo como se puede apreciar en la siguiente figura:

Figura 3.2: Diseño muros

3.3. Muro destructible

Esta clase posee los siguientes atributos:

- x: Posición x
- y: Posición y
- destruido: Atributo de tipo boolean que indica el status del muro durante el juego.Inicialmente es igual a False,debido a que los muros destructibles inicialmente no están destruidos.
- salida: Atributo de tipo boolean, que inicialmente es igual a False. Este atributo indica que muro destructible tiene asociada la puerta de salida.
- rad: Indica el tamaño del objeto para ver si objetos colisionan con él.
- tipo: Atributo que indica de que tipo es el objeto, en este caso, es igual a "MuroDestructible".

Posee el método de dibujar, el cual está creado a partir de cuadriláteros con distintos tonos de iluminación.

Figura 3.3: Diseño muro destructible

3.4. Bombas

Las bombas necesitan los siguientes atributos:

- x: Coordenada en x.
- y: Coordenada en y.
- detonada: Atributo booleano que indica el estatus de la bomba durante el juego. Inicialmente no está detonada, por lo cual es igual a False. Es necesario para indcar si es necesario dibujarla o no en el juego.
- tiempo: Atributo de tipo int que indica el tiempo que se demora la bomba en explotar, despues de ser activada. En este caso, es igual a 3 segundos.
- activada: Atributo booleano que indica el estatus de la bomba. Indica que está activada, por lo tanto, empezará a reducir su tiempo hasta explotar y ser detonada. Inicialmente es igual a False.
- rad: Atributo radio para saber si hay choques.
- tipo: El cual es igual a "Bomba",diferenciando esta clase de las demás.

La clase bombas tiene implementado otro atributo que se llama sonidoExplosion, el cual permite subir un archivo de audio para luego ocuparlo para que la bomba suena cuando explote.

Tiene un método llamado dibujar el cual se compone de primitivas como los poligonos y tambíén con la ayuda de una función circulo que permite dibujar circulos de distintos colores.

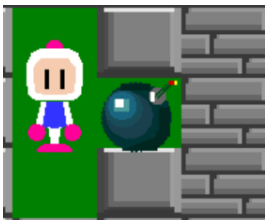

Figura 3.4: Diseño Bomba

La clase bombas posee un método llamado tictac(), el cual va disminuyendo el tiempo de la bomba, restandole un segundo.

Se implementa los métodos llamados explosionCentral y explosionCostados, los cuales en conjunto permiten dibujar la explosión que produce la bomba ya detonada.

Figura 3.5: Diseño explosión

Al llamar al método explosionCentral(), cambia el estatus de la bomba, donde el atributo detonada pasa a ser True, ya que la bomba fue detonada.También en este caso, se realiza a activar el sonido de explosión para que pueda sonar.

3.5. Enemigo

La clase enemigo posee los siguientes atributos:

- x: Posición en x.
- y: Posición en y.
- tipo: El cual es igual a "Personaje"tal como lo es en la clase Personaje.
- amigo: Indica si es amigo o no.En este caso,es igual a False.
- vivo: Indica su estatus en el juego. Inicialmente es igual a True.
- rad: Posee un atributo radio para saber si colisiona con otros objetos
- random: Otorga un número entero entre el 1 y 2, el cual permite elegir que diseño tiene el enemigo.

Posee un método llamado mover, el cual permite mover al enemigo aleatoriamente por el terreno, siempre que le sea posible, ya que no puede pasar si hay bombas en su camino, muros destructibles o muros

Posee un método llamado dibujar que permite dibujar de dos formas, si el atributo random es igual a 1, el enemigo será dibujado como en la figura 3.6 y si no lo es, como la figura 3.7. Ambos están dibujados a través de poligonos y circulos.

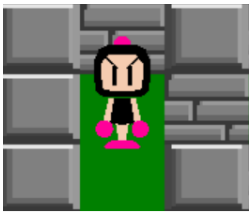

Figura 3.6: Enemigo diseño 1

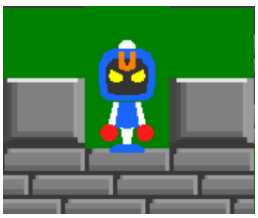

Figura 3.7: Enemigo diseño 2

Por último, como se pide crear enemigos aleatorios y en posiciones aleatorias, se crea el método Aleatorio el cual recibe la lista Personajes y la lista Muros Internos, debido a que el enemigo no se puede crear en la mismas posiciones que estos dos últimos objetos. Se escoje una posición random y se revisa si existe alguna posición que ya se este ocupando por los otros elementos, si no es asi, se puede crear el enemigo sin complicaciones. Este método se ocupa para luego poder crear varios enemigos aleatorios al iniciar el juego.

3.6. Power Up

Se define la clase Power Up con los siguientes atributos:

- x: Indica posición x, la que es igual a la posición x de un muro destructible escogido aleatoriamente antes de comenzar el juego.
- y: Indica posición y, la que es igual a la posición y de un muro destructible escogido aleatoriamente antes de comenzar el juego.
- tipo: Es igual a "PowerUpz su finalidad es poder diferenciar esta clase de las demás.
- rad: Indica el radio del objeto.
- power: Indica que power up tiene, que puede ser igual a "VidaExtra.º "BombaExtra". Inicialmente es igual a ya que aún no se ha definido y posteriormente se definirá de forma aleatoria.
- usado: Indica si el power up se ha usado o no,inicialmente es igual a False.
- recogido: Indica si el power up se ha recogido o no. Inicialmente es igual a False. Si se ha recogido, ya no es necesario dibujarlo en la pantalla, lo que se verá más adelante.
- activado: Indica si el power up ha sido activado, lo que inicialmente es igual a False. Este atributo permite saber cuando se otorgan estos poderes al personaje.
- dibujar: Indica cuando es necesario dibujar el power up, inicialmente es igual a False.

Como se indicó anteriormente, hay dos tipos de power up. El primero es de otorgarle una vida extra al personaje. Para darle más dificultad al juego, se optó por permitir al personaje que tenga como máximo dos vidas, ya que podría recoger varios power ups de vida extra, quitándole la dificultad de morir al juego. El segundo es el de Bomba Extra, el cual le otorga al personaje sólo una bomba extra a poner.Cuando el personaje coloque la segunda bomba, el power up ya habrá sido usado, por lo cual, el personaje vuelve a tener capacidad máxima de poner bombas igual a una, esto quiere decir, que el personaje solo puede poner una bomba a la vez.Luego que exploté la bomba que puso puede volver a poner otra.

Se crearon dos métodos para dibujar los power ups, uno llamado dibujarVidaExtra y el otro dibujarBombaExtra.

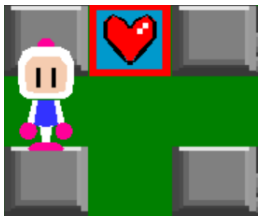

Figura 3.8: Diseño Power Up 1

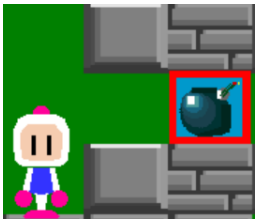

Figura 3.9: Diseño Power Up 2

También se crearon otros dos métodos llamados VidaExtra y otro BombaExtra. El primero modifica el atributo power y lo igual a "VidaExtra".El segundo también modifica el atributo power y lo igual a "BombaExtra". Esto permite diferenciar el power que tiene cada power up.

Como se desean crear power ups aleatorios, por lo cual, se crea el método aleatorio.Este método escoge uno de los dos power up al azar para luego otorgarle una posición de algún muro destructible que no tenga otorgada la salida(esto quiere decir que el atributo salida del muro destructible sea igual a False). Luego se revisa si algunos de los otros power ups tiene la posición creada, si no la tiene, se puede crear el nuevo power up.

Finalmente se crea una función llamada PowerUpsAleatorios, el cual crea power ups aleatorios. Los power ups creados son iguales a la mitad de la cantidad de muros destructibles.

3.7. Reloj

Se crea esta clase con el fin de crear un reloj tipo crónometro, el cual empiece de un tiempo fijo para luego ir disminuyendo segundo por segunda hasta llegar a 0 segundos.
Este reloj tiene el siguiente diseño:

Figura 3.10: Diseño Power Up 2

El único atributo que tiene esta clase se llama tiempo, el cual indica la duración en segundo que tiene el juego, antes de acabar. El tiempo fijado para este juego es de 180 segundos,que es igual a 3 minutos.

Tiene un único método llamado Actualizar, recibe un entero a, el cual si es igual a 0, el reloj no se actualiza. Por otro lado si es igual a 1, el reloj disminuira un segundo su atributo tiempo. Este método será utilizado en el juego cada vez que pase un segundo, ya que el reloj debe cambiar los números. También al llamar al método, este dibujará el reloj en la pantalla de OpenGl.

3.8. Salida

La clase salida posee los siguientes atributos:

- x: Este atributo indica la posición x en el juego, la que será igual a la posición x que un muro destructible aleatorio que se elegirá antes del comienzo del programa.
- y: Este atributo indica la posición y en el juego, la que será igual a la posición y que un muro destructible aleatorio que se elegirá antes del comienzo del programa.
- rad : Permite ver si el objeto tiene colisiones con otros objetos.

Posee el método dibujar, el cual se realiza a partir de primitivas de OpenGL,principalmente de cuadriláteros.

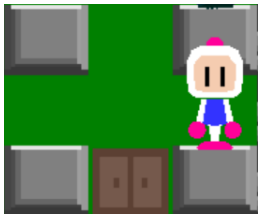

Figura 3.11: Diseño puerta

3.9. Escena

Se crea la clase escena con el fin de definir el espacio donde los personajes podrán interactuar. También para definir las posiciones de los muros internos. La clase tiene los siguientes atributos:

- h: Indica el largo de la ventana.
- w: Indica el ancho de la ventana.
- MurosInternos= Lista que guardará los muros internos del juego.

Esta clase tiene dos métodos. El primero llamado FijarMuros, el cual crea un atributo llamado Muros de tipo lista, que va guardando los muros que contienen distintas posiciones. Esta lista contiene tanto los muros de los bordes que definen el terreno como los muros que van al interior.Los primeros 30 muros de esta lista son los muros internos, por lo tanto, estos se guardan en el atributo MurosInternos.El segundo método es de dibujar, el cual deja un pequeño sector para poder dibujar posteriormente el reloj.El resto del espacio es de color verde en donde se dibujan los muros.

3.10. Vista

La clase vista es implementada para poder dibujar todo lo que se tiene que dibujar según condiciones o los estatus de los objetos. Es por esto que esta clase tiene la función dibujar, la cual recibe los personajes, las bombas, los muros destructibles, la escena (donde están los muros),el reloj,la puerta, los powerups y el tiempo.

Los personajes, las bombas, los muros destructibles y los power ups serán listas con estos objetos para poder analizarlas.

Como se impuso antes, si el personaje que interactua choca con los enemigos, el personaje muere. Para esto, se debe ver si la lista de personajes es mayor a 1, luego se va comparando la posición de cada enemigo con el personaje, si algún enemigo está chocando con el personaje, el estatus vivo=True del personaje cambia a False, por lo tanto, muere.

Para dibujar los muros destructibles se va dibujando uno por uno en la lista de muros destructibles.

Para dibujar los power ups deben cumplir ciertas condiciones para cada power up, las cuales son las siguientes:

- Si primer personaje de la lista de personajes es amigo, se ve si el power up está chocando con él. Si está chocando, cambia el estado recogido del power up a True.
- Se analiza si el power up es de "VidaExtra.º "BombaExtraz si no está activado, si cumple estas condiciones se activan los poderes de cada uno y se cambia el estatus activado del power up a True.
- Si el power up no ha sido recogido y se debe dibujar, se dibuja según que power up sea.
- Finalmente para saber cuando se debe dibujar un power up, debe pensarse que cuando no hay muro destructible en la misma posicion del power up, es porque el muro ya se destruyo, ya que cada power up tiene asociado un muro destructible, es entonces en ese momento cuando el power up se puede dibujar.

La puerta se debe dibujar cuando no hay muro destructible chocando con la puerta, ya que se destruyó el muro que iba asociado a la posición de la puerta.

Las bombas se dibujan una a una y se deben analizar las siguientes condiciones:

- Si una bomba está activada y si el tiempo es mayor o igual a un segunda, se llama a la método tictac(),el cual disminuirá el tiempo de la bomba, restandole un segundo
- Si el tiempo de la bomba es menor o igual a 0, se llama a los métodos explosionCostados y explosionCentral para dibujar la explosion según corresponga y cambie los estatus de la bomba o con lo que la explosión afectó.

Los personajes se deben dibujar solo si se encuentran vivos.

En está sección se dibuja en la pantalla si el personaje gano o perdió. Para el primero, se dibujará "VICTORY"si el primer personaje de la lista personajes es amigo y si el personaje encuentra la salida, ya que no existe otra forma de ganar. Para el segundo, la pantalla mostrará "GAME OVER", cuando el primer personaje no sea amigo,muere personaje amigo, o también cuando se acaba el tiempo del juego.

Por último se realiza un limpiado de la pantalla, eliminando todos los objetos que ya no sirvan. Por ejemplo, cuando las bombas ya son detonadas, cuando los muros destructibles son destruidos, cuando los power ups son usados o cuando los personajes mueren.Esto se realiza a través de una función auxiliar llamada limpiar que verifica el tipo de objeto y el estatus de este para saber si se debe eliminar de la pantalla o no, como también eliminarlo de la lista en la que se encuentra.

3.11. BomberMan Main

Esta es la etapa final en la creación del juego, ya que la vista y el modelo están completos, solo se debe implementar.

Primero se crea la ventana de una altura y ancho determinados para luego inicializar pygame y OpenGl.

Luego se crean las listas de PowerUps, Bombas, Muros Destructibles que inicialmente están vacias. Se crea un objeto vista y se crea la lista de personajes, inicialmente con un personaje amigo en una posición definida. Posteriormente se inicializa la música y los sonidos. Se crea un objeto escena y se fijan los muros de la escena,es decir, se crean.Se crean los enemigos aleatorios agregandolos a la lista personajes,se crean los muros destructibles, se crea y se fija la posición de la salida y finalmente se crean los power ups aleatorios para posteriormente agregarlos a la lista de PowerUps. Por último, se toma el tiempo, para entrar en el loop del programa.Esto se ve reflejado de la linea 52 a la 82 del código que se encuentra en el anexo.

En este loop se va actualizando el tiempo que transcurre.También se analiza los eventos de pygame que ocurren. Por ejemplo, si se presiona la tecla A del teclado, el personaje creará una bomba. Esto pasará solo si no hay otra bomba en juego que aún no haya explotado. También aqui se realizan los movimientos del personaje, para lo cual, se hace una suposición. Por ejemplo, si se apreta la tecla abajo, el personaje debe disminuir de su posición 50, solo si se puede pasar. Por lo cual se usa un personaje auxiliar y se ve si este personaje puede pasar, quiere decir, si no está chocando con murosdestructibles,bombas o muros.

Se actualiza el dibujo con todas las listas de objetos.También si el tiempo es mayor o igual a 1 segundo, el reloj se actualiza, esto quiere decir que disminuye en un segundo su tiempo actual. También todos los enemigos se mueven aleatoriamente y se reinicia el tiempo a 0. Esto beneficia mucho al reloj, debido a que funciona como cronómetro, el cual va disminuyendo un segundo.

También se define cuando debe sonar la música de cuando el personaje gana o cuando el personaje pierde, lo que se puede apreciar entre las lineas 128 y 142. Para el primer caso, es cuando el primer personaje de la lista de personajes no es amigo o cuando el tiempo del reloj es igual a 0, en ese caso, no se actualiza el reloj, el tiempo vuelve a 0 y suena una vez el sonido. Para el segundo

caso, es cuando el primer personaje de la lista de personajes es amigo y si el personaje está chocando con la puerta, es decir, la encontró, es cuando gana, por lo tanto, no se actualiza el reloj y la música suena solo una vez.

Por último, se pone el dibujo en la pantalla.

Es así como el programa realiza este loop infinito hasta que el usuario cierra la ventana o apreta la tecla escape.

Finalmente, luego de terminar la creación del juego, este es el resultado:

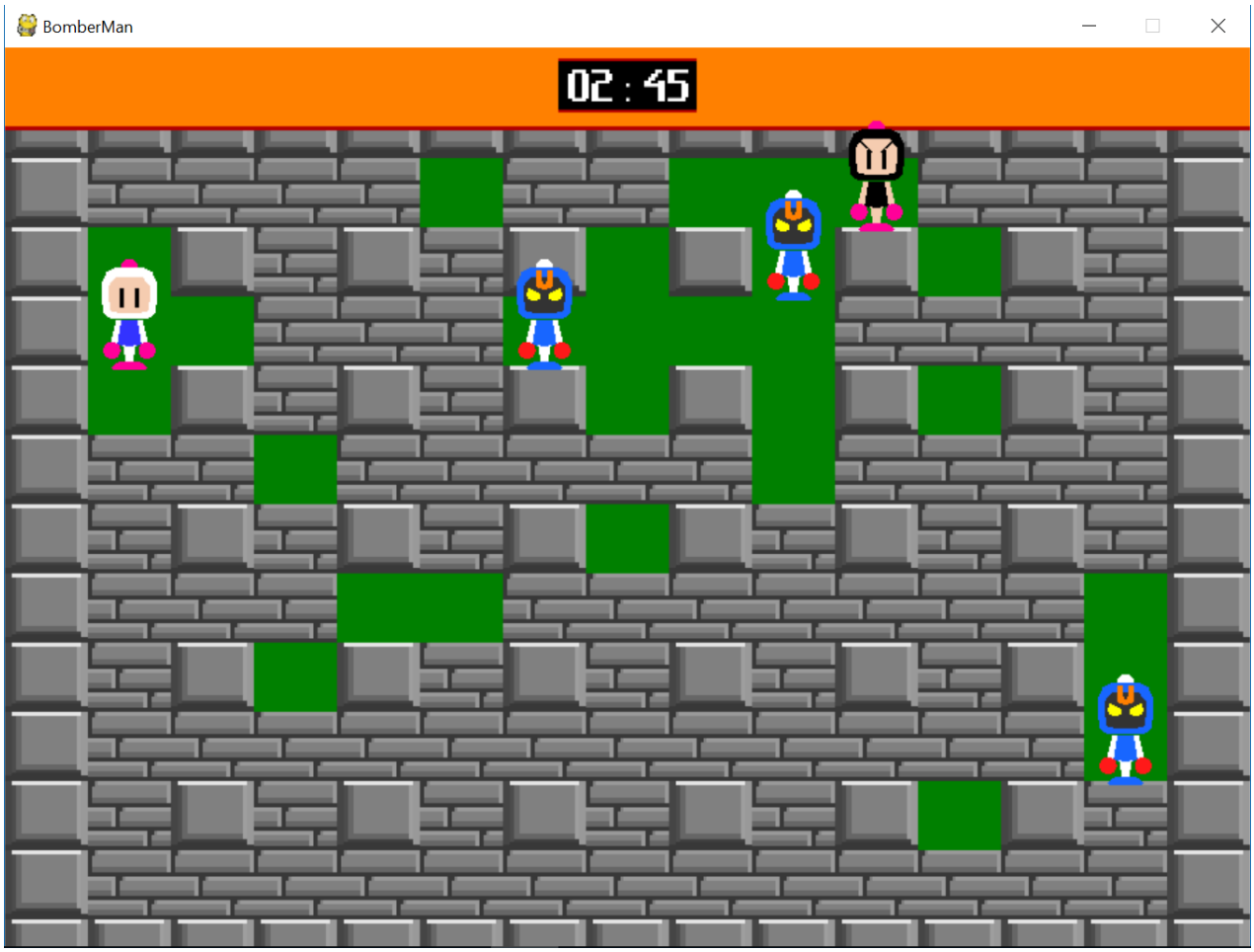

Figura 3.12: Diseño final de BomberMan

4. Discusión sobre dificultades encontradas

Una de las primeras dificultades encontradas al crear el juego, es el diseño de gráficos a partir de primitivas usando el programa OpenGl, debido a que toma bastante tiempo generar personajes u

objetos para luego poder interactuar.Además al generarlos a partir de primitivas,los gráficos quedan de una menor calidad visual,por lo cual,es mucho más fácil y rápido trabajar con sprites e imagenes que ya están completamente definidas y solo se deben usar.

Otra dificultad fue al momento de tener que usar OpenGl y pygame al mismo tiempo,ya que las ventanas se sobreponen. Uno de los casos fue la implementación del reloj, ya que se conocian funciones que ayudan a mostrar texto en la pantalla de pygame, pero que no sirven en OpenGl.Para poder solucionar el problema de imprimir texto en la pantalla de OpenGl, se tuvo que buscar una función auxiliar en Internet que lo hiciera, ya que no existia conocimiento previo.

5. Aprendizajes Obtenidos

El Modelo Vista Controlador que es un patrón de arquitectura de software, orienta y enseña a como organizar las distintas partes de una aplicación.

Para realizar un juego es necesario empezar de cero y desglozar el problema en subproblemas.En este caso, la creación del juego, se subdividió en crear clases para cada objeto que se quisiera representar en la pantalla. Si este procedimiento se hace correctamente, realizar un juego no tomará tanto tiempo.

Pygame permite la creación de videojuegos en dos dimensiones de una manera sencilla y rápida con librerias que ayudan bastante al momento de desarrollar el juego.

Para el diseño gráfico de un juego, se pueden implementar imágenes, que es más fácil y directo al usar pygame.Iambién se pueden diseñar los objetos con primitivas de OpenGl, pero resulta ser más tedioso, ya que reduce la calidad de lo que se quiera representar, por ejemplo una bomba, como también ocupa demasiado tiempo. Otra factor a mencionar es que si se desea dibujar y trabajar con primitivas de OpenGl, es recomendable hacer un dibujo utilizando el programa Geogebra, el cual permite dibujar en 2d, visualizando lo que se desea dibujar y otorgando las coordenadas del objeto. Esto es muy conveniente para obtener un dibujo de mejor calidad visual.

La implementación de sonido en un juego es de suma importancia, ya que da más dinámismo a este y queda más completo.

6. Anexo

```
# -*- coding: iso-8859-1 -*-
import pygame,os,sys
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from Vector2D import *
from Bombas import *
from Enemigo import *
from Escena import *
from Muro import *
from MuroDestructible import *
from Personaje import *
from Vista import *
from Funciones import *
from Reloj import *
from PowerUp import *

#########################################################################
# Funciones de graficos
#########################################################################

def init_pygame(w, h, title=""):
    pygame.init()
    os.environ['SDL_VIDEO_WINDOW_POS'] = str(200) + "," + str(30)
    pygame.display.set_mode((w, h), OPENGL | DOUBLEBUF)
    pygame.display.set_caption(title)

def init():
    glClearColor(0.0, .5, 0.0, 0.0)
    glClearDepth(1.0)
    glDisable(GL_DEPTH_TEST)
    glShadeModel(GL_SMOOTH)
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)
    glEnable(GL_BLEND)
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
    glHint(GL_LINE_SMOOTH_HINT, GL_NICEST)

def reshape(width, height):
    if height == 0:
        height = 1
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(0.0, width, 0.0, height)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

def init_opengl(w, h):
    init()
    reshape(w, h)

```

```
def main(argv):
    w = 900  # ancho
    h = 650  # alto

    PowerUps=[]
    Bombas = []
    MurosDestructibles = []
    Personajes = [Personaje(90, 420),Enemigo(810,170)]
    vista = Vista()
    # inicializando ...
    init_pygame(w, h, "BomberMan")
    init_opengl(w, h)

    # música de fondo
    pygame.mixer.music.load("musica/BomberMan.mid")
    pygame.mixer.music.play(-1, 0.0)
    #sonidos
    sonidoMuerte= pygame.mixer.Sound("musica/Gameover.wav")
    sonidoGana = pygame.mixer.Sound("musica/Victory.wav")
    sonido=True

    es = Escena(w, h)
    es.FijarMuros()

    CrearEnemigos(Personajes,es.MurosInternos)
    CrearMurosDestructibles(Personajes,es.MurosInternos,MurosDestructibles
   )
    Puerta=Salida(PosicionSalidaAleatoria(MurosDestructibles))
    PowerUpsAleatorios(PowerUps,MurosDestructibles)
    reloj=Reloj()
    run = True
    t = 0
    t0 = pygame.time.get_ticks()
    while run:
        t1 = pygame.time.get_ticks()  # tiempo actual
        dt = (t1 - t0)  # diferencial de tiempo asociado a la iteración
        t0 = t1  # actualizar tiempo inicial para siguiente iteración

        for event in pygame.event.get():
            # cerrar
            if event.type == QUIT:
                run = False

            if event.type == KEYDOWN:
                if event.key == K_RIGHT:
                    auxpersonaje = Personaje(Personajes[0].x+60,Personajes
   [0].y)
                    if PuedoPasar(auxpersonaje,Bombas,MurosDestructibles,
   es):
                        Personajes[0].x+=60
                elif event.key == K_LEFT:
                    auxpersonaje = Personaje(Personajes[0].x-60,Personajes
```

```
    [0].y)
100                 if PuedoPasar(auxpersonaje, Bombas,MurosDestructibles,
    es):
101                     Personajes[0].x-=60
102             elif event.key == K_DOWN:
103                 auxpersonaje = Personaje(Personajes[0].x,Personajes
    [0].y-50)
104                 if PuedoPasar(auxpersonaje, Bombas, MurosDestructibles
    ,es):
105                     Personajes[0].y-=50
106             elif event.key == K_UP:
107                 auxpersonaje = Personaje(Personajes[0].x,50+Personajes
    [0].y)
108                 if PuedoPasar(auxpersonaje, Bombas, MurosDestructibles
    ,es):
109                     Personajes[0].y+=50
110             elif event.key == K_a:
111                 if(Personajes[0].bombas>=1):
112                     NuevaBomba=Bomba(Personajes[0].x,Personajes[0].y)
113                     NuevaBomba.activada=True
114                     Bombas.append(NuevaBomba)
115                     Personajes[0].bombas -= 1
116
117             # cerrara
118             elif event.key == K_ESCAPE:
119                 run = False
120
121
122     t = t + dt
123     vista.dibujar(Personajes,Bombas,MurosDestructibles,es,reloj,t,
    Puerta,PowerUps)
124     if t>=1000:#tiempo sea un segundo
125         reloj.Actualizar(1)
126         MoverEnemigos(Personajes, Bombas, MurosDestructibles, es)
127         t=0
128     if not Personajes[0].amigo or reloj.tiempo==0:
129         reloj.Actualizar(0)
130         t=0
131         if sonido:
132             sonido=False
133             pygame.mixer.music.stop()
134             sonidoMuerte.play()
135     elif Personajes[0].amigo :
136         if Personajes[0].EncuentraSalida(Puerta):
137             reloj.Actualizar(0)
138             t=0
139             if sonido:
140                 sonido = False
141                 pygame.mixer.music.stop()
142                 sonidoGana.play()
143     # pone el dibujo en la pantalla
144     pygame.display.flip()
```

```
145         # ajusta para trabajar a 30 fps.
146         pygame.time.wait(int(1000/30))
147 
148     # termina pygame (cerrar ventana)
149     pygame.quit()
150 
151 if __name__ == "__main__":
152     import sys
153     main(sys.argv)
```

Lista de Códigos Fuente 1: BomberMan