Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

Tarea 2

CC3502 – Modelamiento y Computación Gráfica para Ingenieros

Nombre:	Sebastián Parra
Profesora:	Nancy Hitschfeld
Auxiliares:	Pablo Polanco
	Pablo Pizarro
	Mauricio Araneda
Ayudantes:	Iván Torres
	María José Trujillo
Fecha:	8 de julio de 2018

Descripción del Problema

La tarea consistió en programar una recreación del videojuego Bomberman. El juego trata sobre las aventuras de un hombre-robot que debe atravesar un laberinto mientras evita diversos enemigos. Para esto, Bomberman cuenta con un arsenal de bombas, cuyas explosiones lo pueden ayudar a borrar obstáculos del mapa o eliminar a sus enemigos.

El problema a resolver se dividió en 3 partes: El diseño de gráficos, que consistió en el diseño e implementación en pantalla de todos los componentes del juego (personaje principal, enemigos, escenario, etc.), seguido de la elaboración de una mecánica simple de juego, en donde se programaron las interacciones principales entre los componentes anteriormente creados, y finalmente la implementación de características avanzadas, donde se dedicó a programar otras características más complejas del juego, cuya implementación no necesariamente obliga a mantener las funcionalidades anteriores del programa.

Esquema de la Solución

Para el desarrollo del juego se utilizó el modelo-vista-controlador visto en clases, reconociéndose las siguientes funciones y elementos principales:

Modelo:

- Personaje Principal
- Muros (destructibles e indestructibles)
- Bombas
- *Power Ups* (2 diseños distintos)
- Enemigos (2 diseños distintos)
- Escena

Vista:

- Ventana: Encargada de dibujar todos los elementos del modelo

Controlador:

- Inicialización del juego (distribución de los elementos en la pantalla)
- Comportamiento de enemigos (movimiento, colisiones, e interacciones con el jugador)
- Comportamiento de las bombas (revisar si la bomba debe explotar y dibujar su estado correspondiente, interacciones con jugador, enemigos y muros destructibles)
- Manejo de eventos (colocar una bomba)
- Movimiento del jugador y colisiones

Explicación Detallada de la Solución

Diseño de Gráficos

Para esta sección se reconocieron los siguientes elementos:

1. **Personaje Principal:**

   Como esta figura se debía implementar únicamente con primitivas de OpenGL y corresponde a uno de los componentes con mayor número de interacciones (colisiones) con otros elementos del juego,

se decidió programar la figura de un robot utilizando solamente rectángulos, como se puede apreciar en la figura 1.

Otro factor que se consideró fue que el jugador debe poder chocar con sus propias bombas, por lo que sería deseable que las bombas aparecieran frente al jugador, en vez de directamente sobre éste. Por esta razón, también se implementaron 3 dibujos más para el personaje principal, mostrándolo mirando hacia las 4 direcciones posibles de movimiento, como se puede apreciar en la figura 2.

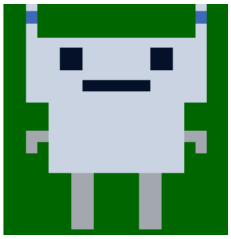


Figura 1: Personaje principal.

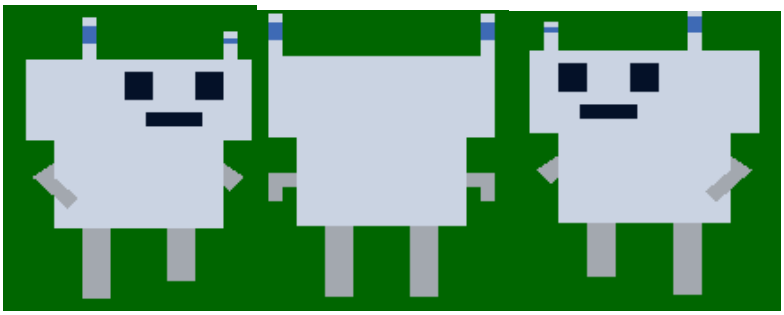
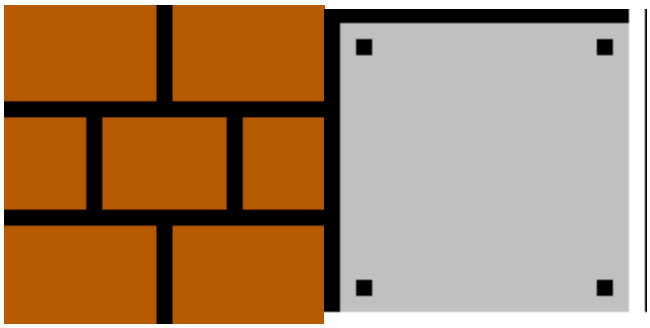


Figura 2: Distintas caras del personaje principal.

2. **Muros:**

Se programaron dos diseños: Uno para muros destructibles (figura 3a) y otro para indestructibles (figura 3b). Dada la simetría de los diseños, estos bloques simplemente se ubicaron uno al lado del otro para simular la idea de muros más grandes.



a) b)

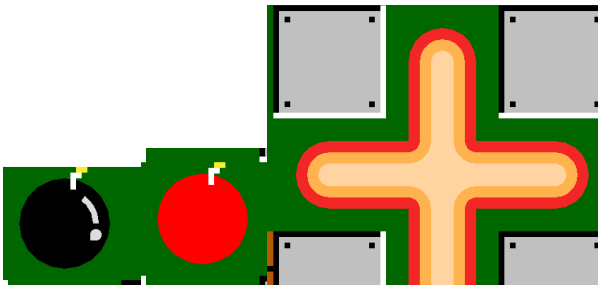
Figura 3: Dibujos implementados para los muros a) destructibles, y b) indestructibles.

3. **Bombas:**

Las bombas se dibujaron como un círculo en negro, con una mecha blanca y una franja gris para dar el efecto de brillo. En cuanto a la explosión, ésta se dibujó utilizando rectángulos y una semi circunferencia para las “puntas” de los pilares.

Adicionalmente, se dibujó una segunda bomba con las mismas dimensiones, pero pintada de rojo, para alertar al jugador que la bomba va a explotar pronto mediante parpadeos de color.

En la figura 4 se pueden apreciar todos los estados de la bomba.



a) b) c)

Figura 4: Dibujos para los distintos estados de la bomba.

4. **Power Ups:**

Como se verá posteriormente al mostrar una escena estática del juego, se programaron dos diseños distintos: Uno para aumentar el alcance explosivo de las bombas, y otro para aumentar el número de bombas simultáneas que puede poner el jugador. Los diseños dibujados fueron inspirados por los utilizados para el videojuego original de Bomberman.

5. **Enemigos:**

Se realizaron dos diseños distintos de enemigos (los cuales también se podrán observar en la escena estática), basados principalmente en la utilización de rectángulos y fragmentos de circunferencias, con el objetivo de simplificar lo más posible el cálculo de colisiones.

6. **Escena:**

Finalmente, en la figura 5 se puede observar una escena estática del juego, donde se muestran todos los elementos mencionados en esta sección.

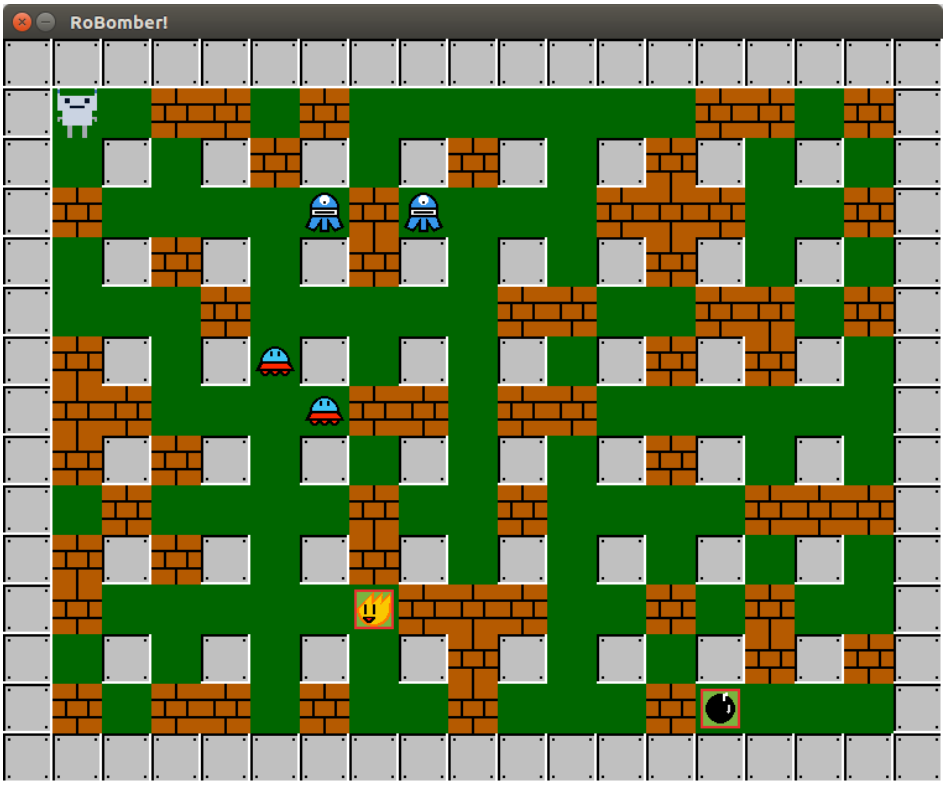


Figura 5: Escena estática del juego.

Mecánica Simple de Juego

1. **Movimiento del personaje:**

Para implementar esta funcionalidad se utilizó la función de pygame *key.get_pressed()* para poder implementar el movimiento del jugador al presionar las teclas arriba, abajo, izquierda, y derecha, junto con la implementación de manejo de eventos utilizando las rutinas ofrecidas por pygame.event para la colocación de bombas. La mayor ventaja de haber utilizado la primera función para el desplazamiento fue que permitió que el jugador se moviera al dejar presionada la tecla, sin necesidad de tener que levantarla. Además, en estas mismas secciones de código de verificaron las colisiones con muros y bombas, lo cual se realizó mediante la superposición de un rectángulo de pygame (objeto pygame.Rect) con cada elemento del juego, y así poder utilizar la función colliderect() de esta librería, simplificando una gran cantidad de líneas de código.

2. **Rutina de explosión de las bombas:**

Para esta mecánica se implementó un temporizador como atributo de la clase Bomba, el cual, luego de haber puesto una bomba, disminuye en 1 su valor para cada recuadro, pudiendo así fijar el tiempo de explosión de las bombas en 3 segundos al inicializar el temporizador en 90, considerando que el juego se refresca a 30 *frames* por segundo. Luego de esto, la programación de las explosiones se redujo simplemente a cambiar la forma de dibujar el objeto según el valor que tenga su temporizador.

3. **Término del juego:**

Esta funcionalidad se programó mediante la función colliderect() de pygame, donde si un enemigo o una bomba en estado de explosión colisiona con el jugador, el juego se congela y se imprime en consola el string “GAME OVER”.

4. **Eliminación de muros destructibles y enemigos:**

En el caso de la eliminación de muros destructibles, surgió como desafío el pensar cómo encontrar los muros destructibles que se encuentran dentro del rango de una explosión. Para esto, se programó una matriz que guardase la ubicación (en notación (fila, columna)) de todos los obstáculos en el laberinto, junto con un diccionario que, a partir de los índices de un obstáculo que se encuentra en la matriz, indica la ubicación del objeto correspondiente en la lista de objetos a dibujar. Por ejemplo, si en un cierto momento existen 20 muros destructibles en el juego, y se coloca una bomba cerca de uno de estos muros, al introducir la ubicación (fila, columna) de este obstáculo en el diccionario, se podrá descifrar cuál de los 20 objetos almacenados en la lista corresponde al muro cuyas colisiones desean evaluar. Una vez implementadas estas estructuras, sólo restó usar la función colliderect() de pygame para ver si las *hitboxes* se intersectaban.

En cuanto a la eliminación de enemigos, como el juego sólo posee 4 enemigos como máximo, se decidió simplemente que cada vez que exista una bomba en estado de explosión, se probarán colisiones con todos los enemigos en la lista de enemigos a dibujar.

Finalmente, para ubicar la salida simplemente se buscaron las posiciones de la matriz de obstáculos que contenían un muro destructible, y se eligió una coordenada aleatoria entre éstas para ubicar este elemento.

Características Avanzadas

1. **Aparición aleatoria de los muros, *power ups*, enemigos, y la salida:**

La solución implementada se basó en la utilización de la matriz de posiciones que indica si un objeto se encuentra en una determinada celda o no (la codificación utilizada fue cero para celdas sin objeto, 1 para celdas con un muro indestructible, 2 para celdas con un muro destructible, 3 para celdas con bombas, y 4 para celdas con enemigos). De esta manera, las ubicaciones aleatorias se pueden obtener rellenando la matriz con las ubicaciones de los muros indestructibles, luego retirando todas las coordenadas que podrían ser candidatos de contener un muro (celdas sin ningún obstáculo), un *power up* (celdas con un muro destructible), enemigo (celdas sin obstáculos), o una salida (celdas con un muro destructible), y finalmente tomando una cantidad de muestras aleatorias, igual a la cantidad de objetos que se desean dibujar, de estos candidatos. Sin embargo, surge inmediatamente la problemática de que al seguir esta metodología existe la probabilidad no nula de que el juego se inicialice en condiciones imposibles de jugar (por ejemplo, un muro sobre el jugador, el jugador acorralado por muros, la salida encima del jugador, etc.), por lo que al realizar este procedimiento se optó por no considerar las celdas inmediatamente vecinas al personaje principal.

2. **Movimiento aleatorio de los enemigos.**

Para implementar está característica, primero se fijó una velocidad de movimiento tal que cada 20 recuadros, el enemigo se haya trasladado exactamente una celda, junto con un temporizador para

cada enemigo, el cual comienza en 20 y disminuye en 1 su valor cada recuadro, para así saber cuándo un enemigo en particular ha avanzado la distancia de una celda.

Teniendo esto en mente, se implementó la siguiente rutina: Cuando se inicializa el enemigo (con su temporizador en 20), se calcula la celda (en notación (fila, columna)) en la que se encuentra la mayoría del cuerpo del objeto, para así poder obtener todas las direcciones posibles de movimiento del enemigo en donde no hay obstáculos, por medio de la utilización de la matriz de posiciones. Una vez obtenido el listado de todas las direcciones posibles, se elige aleatoriamente una dirección de movimiento. Luego, el enemigo comienza a moverse en esa dirección hasta que su temporizador llegue a cero, o bien hasta que se encuentre con otro enemigo durante su movimiento. Si ocurre el primer caso, simplemente se reinicia el temporizador y se escoge una nueva dirección de movimiento aleatoria, mientras que, si ocurre el segundo caso, se cambia de sentido el movimiento del enemigo y se mantiene ese desplazamiento hasta que su temporizador llegue a cero, o hasta que el enemigo vuelva a alinearse con una celda (ya que, dependiendo del instante de tiempo en que ocurra la colusión entre enemigos, la alineación puede perderse).

3. **Implementación de los *power ups:***

En el videojuego que se desarrolló, existen dos powerups distintos, cuya funcionalidad se explicará en detalle a continuación:

- **Mayor alcance de explosión:**

  Para implementar este power up, se fijó una variable en el controlador que maneja el alcance que poseen todas las bombas a crear (se inicializa en 1). Luego, este valor se utiliza al dibujar los pilares de la explosión de las bombas, cuya extensión se escala según el alcance de la bomba.

  Sin embargo, al entrar en juego este power up surge el problema de detectar cuando una explosión colisiona con un muro, ya que, si este es el caso, el alcance del pilar que colisiona no debe extenderse más allá de la celda en la que se ubica el muro, lo cual debe ocurrir sin modificar los alcances de los otros pilares. Para solucionar este problema, se dividió el alcance de cada bomba en las cuatro direcciones en que se extiende: Norte, sur, este y oeste, para así tener una mayor libertado en cuanto a qué pilar debe ver modificado su alcance.

- **Mayor capacidad para colocar bombas:**

  Este powerup simplemente aumenta la cantidad de bombas simultáneas que puede colocar el jugador (inicialmente este valor es 1). Para esto, se implementó una variable en el controlador que limita el número de bombas que pueden estar activas en el juego simultáneamente. Así, cada vez que el jugador desea crear una bomba, se revisa si el largo de la lista de bombas a dibujar es menor al límite de bombas impuesto, y sólo si esto se cumple el jugador podrá colocar una bomba. Luego, si hay bombas en juego y éstas explotan, éstas son borradas de la lista de bombas después de ser borradas de la pantalla. De esta forma, la implementación permite que el efecto del power up pueda apilarse todas las veces que se desee.

Dificultades Encontradas

La primera dificultad que se encontró resultó ser la más importante de ellas, la cual fue familiarizarse con las librerías de pygame y OpenGL, junto con la metodología de programación por modelo-vista-controlador. Resultó que, durante las primeras horas de trabajo, se invirtió mucho tiempo tratando de entender el funcionamiento de las librerías, observando ejemplos subidos en el material del curso, buscando tutoriales, y pensando en qué sección del programa comenzar, ya que la tarea parecía tener tantos componentes que pensar en una solución para todos los objetivos al mismo tiempo era una tarea abrumadora. Sin embargo, una vez superada esta barrera de entrada, la metodología de modelo-vista-controlador logró facilitar en gran medida la tarea de aislar los distintos problemas a solucionar, pudiendo separar de buena manera la realización de los diseños gráficos, el dibujo de éstos en pantalla, y el manejo de las interacciones entre los elementos del juego.

Otra dificultad que destacó entre todas las encontradas fue la discrepancia entre los sistemas coordenados de OpenGL y pygame, puesto que la primera librería ubica su origen en el lugar esperado para un sistema cartesiano, que es la esquina inferior izquierda de la pantalla, mientras que pygame ubica su origen inspirándose en las convenciones históricas adaptadas para los sistemas raster-scan, que es la esquina superior izquierda de la pantalla. Esta discrepancia entre sistemas no generó más que un sin fin de errores y complicaciones al tratar de coordinar objetos de una librería con la otra. Por ejemplo, como se utilizaron objetos de pygame para modelar los rectángulos de colisión de los elementos del juego, cada vez que se movía un objeto, el rectángulo también debía moverse, pero adaptándose al sistema de coordenadas de pygame. En resumen, esta discrepancia resultó ser el origen de la mayoría de los errores encontrados durante la programación del controlador.

Aprendizajes Obtenidos

A través de esta tarea se pudo tener un grado mayor de acercamiento a la utilización de OpenGL, que es una de las librerías de computación gráfica más utilizadas, para programación en 2D, además de conocer un poco sobre librerías anexas que expanden las funcionalidades de OpenGL para aplicaciones específicas, como lo es pygame con el desarrollo de videojuegos simples. Estos conocimientos pueden servir a futuro al ayudar a familiarizarse con conceptos que son comunes a todas estas librerías, como lo son las rutinas de dibujo, actualización, y borrado de las pantallas. De esta manera, se logra obtener un piso base de conocimiento de librerías de este tipo, para así lograr mitigar el problema de la barrera de entrada mencionado anteriormente.

Otro aspecto importante de esta tarea fue la familiarización con el paradigma MVC visto en clases, a un nivel más práctico, donde a lo largo de la tarea se pudo evidenciar la capacidad de esta metodología de dividir problemas complejos en problemás más simples y pequeños.