TEC | Tecnológico de Costa Rica

Área académica de ingeniería en computadores

Lenguajes, Compiladores e Intérpretes | CE3104

Grupo 1

Tarea 3: Programación imperativa y OO

BreakOutTec

Profesor: Marco Rivera Meneses

Estudiantes:

- Jose Antonio Espinoza Chaves - 2019083698
- Eduardo Zumbado Granados – 2019003973

I Semestre 2022

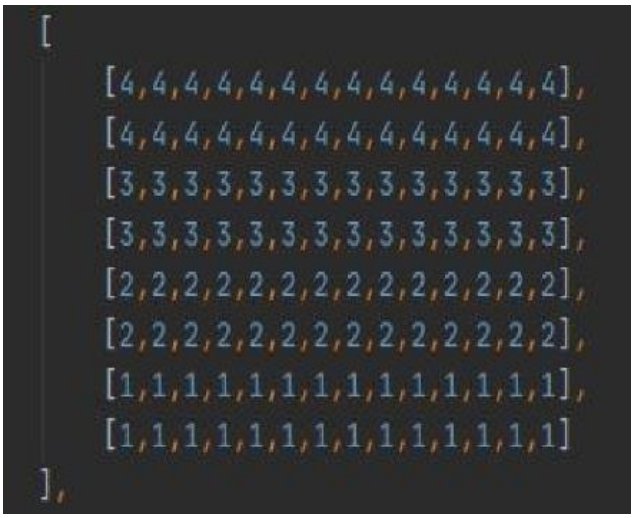
Índice

Descripción de la utilización de las estructuras de datos desarrolladas

En la estructura del juego se utilizó un arreglo bidimensional el cual tiene 8 filas y 14 columnas, estas entradas representan los diferentes bloques que puede haber en el juego, dicha está contenida dentro de un struct junto con las demás variables del juego. Esta estructura es utilizada para la lectura tanto en el cliente como en el servidor.

En la matriz se utilizan diferentes números para representar los diferentes bloques:

0. No hay bloque en ese espacio
1. Bloque verde
2. Bloque amarillo
3. Bloque naranja
4. Bloque rojo
5. Bloque que da una vida
6. Bloque que aumenta la cantidad de vidas
7. Bloque de aumento de velocidad
8. Bloque que reduce la velocidad
9. Bloque que aumenta el tamaño de la raqueta
10. Bloque que reduce el tamaño de la raqueta.



Como se puede observar, el arreglo bidimensional cuenta con dos filas de bloque verde, dos filas de bloque amarillo, dos filas de bloque naranja y dos filas de bloque rojo. Esta cambiará de valor al ser destruida o activada en el cliente. Esta estructura también cuenta con miembros como el número de vías, el puntaje, cantidad de bolas, velocidad de la bola, tamaño de la raqueta y su ubicación.

En el cliente se utilizaron dos estructuras de datos principales. La primera es una matriz de la forma Integer[][], la cual se actualiza con los valores provenientes del servidor y es atravesada para conseguir los valores necesarios para actualizar los valores físicos del cliente. La otra estructura de datos utilizada es el ArrayList<>, el cual se utilizó para manejar la lista de bolas y la lista de bloques.

Ahora, para la implementación de los bloques, se utilizó un patrón de Factory, donde la clase creadora declara los métodos fábrica que devuelve nuevos objetos de bloque. De esta manera, se le aplica como creador la clase abstracta de Brick.java y su salida es cualquiera de los tipos de bloques dependiendo de su entrada.

```java
public class BrickFactory {
	/**
	 * Function that fabricates a brick object
	 * @param type type of brick
	 * @param x x pos in the UI
	 * @param y y pos in the UI
	 * @param w width
	 * @param h height
	 * @param points points that the brick gives
	 * @param color color of the brick
	 * @return a brick object with the given values.
	 */
	public Brick getBrick(BrickType type, Integer x, Integer y, Integer w, Integer h, Integer points, Color color) {

		switch(type) {
			case NORMAL:
				return new NormalBrick(x, y, w, h, points, color);
			case LIFE:
				return new LiveBrick( x,  y,  w,  h,  points,  color);
			case BALL:
				return new BallBrick( x,  y,  w,  h,  points,  color);
			case DOUBLESIZE:
				return new RacketDoubleSizeBrick( x,  y,  w,  h,  points,  color);
			case MIDSIZE:
				return new RacketMidSizeBrick( x,  y,  w,  h,  points,  color);
			case INCVEL:
				return new IncreaseVelBrick( x,  y,  w,  h,  points,  color);
			case DECVEL:
				return new DecreaseVelBrick( x,  y,  w,  h,  points,  color);
		}

		return null;

	}

}
```

Descripción detallada de los algoritmos desarrollados

Servidor

createGame(int greenVal, int yellowVal, int orangeVal, int redVal, double newBallSpeed, int levelup, int newScore):

-Inicializa por defecto todos los miembros del struct “game”.

levelFinished():

-Revisa la matriz de bloques, para ver si está vacía, se utiliza como un flag.

nextLevel():

-Revisa si el nivel ha terminado, de ser así, vuelve a iniciar el juego con los nuevos valores correspondientes.

jsonGame(char* p):

-Se encarga de convertir la estructura actual del juego en formato de una cadena tipo JSON. Para esto, utiliza funcionalidades de las librerías incluidas en el punto cinco y seis de la bibliografía, insertando en la cadena los valores de cada uno de los miembros de la estructura principal del juego.

jsonParse():

-Se encarga de leer un archivo JSON.

useJsonFile(char* newJson):

-Se encarga de ingresar el string json en un archivo .json

updateScore(int brickID):

-Actualiza el puntaje del jugador.

updateGameStats(int brickID):

-Se encarga de actualizar los valores del jugador, vidas y tamaño de raqueta.

receiveClientMessage(char* msg):

-Es un algoritmo para la interpretación del mensaje recibido en el servidor desde el cliente para así modificar la estructura del juego. Se encarga de dividir el mensaje de la siguiente manera: la primera palabra corresponde con el evento que sucedió en el cliente, por ejemplo “Broke”. Lo que le sigue son las coordenadas o índices en la matriz del bloque que se rompió. Por lo tanto, un mensaje válido sería “Broke 1 2”. Si el mensaje no es válido, este es omitido.

Una vez el mensaje fue analizado, se procede con la modificación de la estructura del juego, si un ladrillo se quebró, se realizan las siguientes acciones:

a. Se busca en la matriz según las coordenadas o índices.
b. Se pone el valor de ese ladrillo en cero, para indicar que fue roto.
c. Se analiza el número que indica el tipo de ladrillo y se realiza la acción correspondiente, ya sea simplemente aumentar el puntaje o dar una vida extra, bola extra, etc.

receiveUserMessage(char* msg):

-Algoritmo que analiza el mensaje del usuario desde consola para así modificar la estructura del juego. Obtiene y divide el mensaje. Analiza las coordenadas dependiendo del valor de estas, se coloca el número identificador del ladrillo que el usuario desea poner en ese lugar. De ese modo, el cliente reciba la matriz modificada y se muestra en la pantalla.

Cliente

createContentGame():

-Crea el contenido de la ventana, contiene un timer que se comporta como un main loop del juego y llama a Update() para actualizar la ventana.

createContentMenu():

-Crea el contenido de la ventana del menú, los botones iniciales.

Update():

-Actualiza todos los elementos en la ventana actual, se utiliza para mostrar al cliente los valores actualizados de vidas, puntuación y nivel.

createMatrix():

-Se encarga de crear la matriz de bloques de dimensión 8x14.

printMat(Integer [][] mat):

-Se encarga de recorrer la matriz e imprime sus valores en consola.

checkMatrixChange():

-Método booleano, revisa si hay cambios en la matriz del JSON, y hace los cambios correspondientes en la GUI.

clearBalls():

-Limpia la lista de bolas, y destruye los widgets de la bola en la GUI.

clearBricks():

-Limpia la lista de bloques, y destruye los widgets correspondientes.

spawnBall(Integer quantity):

-Agrega una bola al juego, y se dibuja en la GUI.

brickAction(String action, Integer points):

-Cambia las variables del juego cuando se rompe un bloque. Tiene varios casos dependiendo del tipo de bloque que entra como “action” y dependiendo de este, se ejecuta una acción.

Problemas sin solución

- Se presentó un problema, el cual no se pudo solucionar, relacionado con la conexión mediante sockets del cliente y el servidor. A la hora de enviar mensajes de vuelta al cliente desde el servidor, este no se logra enviar de inmediato con la matriz actualizada. En otras palabras, existe un delay o retraso en la lectura de esta matriz, por lo que el cliente no puede

mostrar en cada momento el juego actual, aunque el servidor sí se encuentra la matriz actualizada.

- No se logró implementar el cliente espectador.

Plan de Actividades realizadas por estudiante

Estudiante	Fecha	Actividad
Ambos	02/06/2022	Crear el repositorio del proyecto
Ambos	02/06/2022	Leer la especificación y repartir tareas
Jose	04/06/2022	Iniciar con la programación de la UI
Eduardo	05/06/2022	Plantear la estructura de OOP para el cliente
Eduardo	08/06/2022	Crear parser de Json Files en el cliente Java
Jose	08/06/2022	Programar los aspectos de OOP de la bola y de la raqueta, así como los stats del juego
Eduardo	08/06/2022	Crear el proyecto del servidor, usando el IDE Clion, además agregar las librerías.
Ambos	10/06/2022	Crear los struct para el servidor, así como las variables necesarias.
Eduardo	12/06/2022	Crear el socket que conecte ambos programas mediante archivos en formato json.

Problemas encontrados

- En la conexión entre el servidor (C) y el cliente (Java), se notó un problema que, a la entrega de este proyecto, no se pudo solucionar. El problema consiste en la lectura errónea por parte del servidor de los mensajes enviados por el cliente, ya que el servidor lee y registra dos mensajes separados enviados por el cliente, el primer mensaje trata del primer carácter del dato enviado por el cliente, mientras que el segundo mensaje consta del resto del dato enviado por el cliente.

  Ya que no se pudo saber la causa de este error, se procedió a manejarlo insertando un carácter predeterminado al inicio de todos los mensajes enviados por el cliente. Este carácter se decidió que fuera un slash (/) , de esta manera el servidor reciba el primer carácter (/), mientras que el segundo carácter es el mensaje completo del cliente. De esta manera se puede descartar el primer carácter y trabajar con el segundo. Por este proceso es que se plantera que se da el error antes mencionado de lectura errónea del Json.

Conclusiones del Proyecto

Luego del desarrollo del proyecto, se observó que implementar una solución combinada de servidor en C y UI en Java es viable, pues permite utilizar lo mejor de cada lenguaje, como por ejemplo en administración de memoria con C y desarrollo de UI con Java. Además, permite combinar el paradigma imperativo con el orientado a objetos.

Además, se demostró facilidad para compartir información entre dos lenguajes diferentes, mediante el uso de socket y de un formato estandarizado, como lo es json, esto permitió una comunicación optima entre ambos sistemas.

Se logró modelar el problema mediante la implementación de diversas estructuras y clases, como listas, estructuras que pueden ser fácilmente utilizadas en ambos lenguajes. Por lo anterior, se concluye que para el desarrollo de un juego como breakOutTec, la utilización de los paradigmas imperativo y orientado a objetos es conveniente.

Recomendaciones del Proyecto

Se recomienda la utilización del paradigma y orientado a objetos e imperativo. Esto pues el problema puede ser modelado y solucionado de manera natural utilizando ambos paradigmas.

También, se recomienda la utilización de un API para la conexión e intercambio de información entre Java y C, esto pues, los sockets son estructuras fuertemente dependientes de hilos y si estos reciben muchas entradas al mismo tiempo pueden cerrar el socket, lo cual echa a perder la conexión.

Por último, se recomienda que, al trabajar en el sistema operativo de Windows, ciertos aspectos en la parte de conexión se tienen que tomar en cuenta, por ejemplo, el uso de la librería WinSock y el enlace que se debe hacer con ws2_32, esto se realiza mediante la bandera “-lws2_32” que se debe colocar en el tiempo de compilación del servidor. Debido a problemas con la conexión en el ambiente de Linux, se recomienda usar el juego en ambiente de desarrollo de Windows.

Bibliografía consultada en todo el Proyecto

[1] Microsoft, “CreateThread function (processthreadsapi.h)”, 2021. [Online]. Disponible en: https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread

[2] TutorialsPoint, “C - Input and Output”, 2020. [Online]. Disponible en: https://www.tutorialspoint.com/cprogramming/c_input_output.htm

[3] GeeksforGeeks, “Command line arguments in C/C++”, 2022. [Online]. Disponible en: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/

[4] StackOverflow, “C split a char array into different variables”, 2012. [Online]. Disponible en: https://stackoverflow.com/questions/10349270/c-split-a-char-array-into-different-variables

[5] Github/rafagafe, “JSON Maker”, 2021. [Online]. Repositorio disponible en: https://github.com/rafagafe/json-maker

[6] Github/tiny-json, “tiny-json”, 2021. [Online]. Repositorio disponible en: https://github.com/rafagafe/tiny-json

Enlace del repositorio

https://github.com/JoseA4718/BreakOutTEC