Metodología de Diseño – Proyecto

Integrantes: Tomás González M.

Fecha: 30 de Abril de 2019.

Objetivo

Documentar los productos de los desarrollos del problema1 utilizando diagramas UML de diseño: Diagrama de clases, diagrama de secuencia. Además, contemplando breves descripciones sobre el problema original y la solución implementada aplicando patrones de diseño.

1. Contexto

Como manera de introducir, es necesario saber y conocer los patrones principales a explicar en el desarrollo de este informe, como lo son el GOF y la forma de programación conocida como MVC que serán descritas y explicadas de manera breve a continuación. Patrones de Diseño GOF.

¿Qué es?

El patrón de diseño Estrategia es uno de los patrones de diseño conocidos como GoF (de Gang of Four, por sus cuatro creadores: Gamma, Helm, Johnson y Vlissides). Son patrones de **diseño orientado a objetos**, que encapsulan soluciones a problemas comunes de diseño.

Existen tres tipos de patrones de diseño GoF, que resuelven tres tipos de problema distintos:

- **Creacionales**, que resuelven problemas de creación de objetos, como, por ejemplo, ¿cómo crear un objeto si no sé su clase? ¿cómo crear un objeto idéntico a otro? ¿cómo crear un objeto de la misma clase de otro, pero sin saber su clase?
- **Estructurales**, que resuelven problemas para organizar objetos en estructuras dinámicas, recursivas o en contextos de ambigüedad, como por ejemplo ¿Cómo represento un árbol?, ¿cómo le puedo agregar atributos a una clase en tiempo de ejecución? ¿cómo represento una estructura recursiva de todo y partes?
- **De Comportamiento,** que resuelven problemas de comportamiento dinámico o en condiciones de ambigüedad, como por ejemplo ¿Cómo hago que un objeto cambie su comportamiento para un mismo método en tiempo de ejecución? (Como el patrón estrategia)

MVC.

¿Qué es?

El Model-View-Controller representa una forma de programación de resolución de requerimientos a partir de problemas conocidos o planteados. A partir de esto, puede ser generado un **diagrama de secuencia** que explique de manera gráfica los procesos e interacciones entre el modelo, la vista y el controlador

2. Problema

Problema 1:

Descripción: A partir de diversos juegos proporcionados, fue escogido el juego denominado “Catch”, el cual consiste en una barra en posición horizontal que impide el paso de pelotas que bajan verticalmente de arriba hacia abajo.

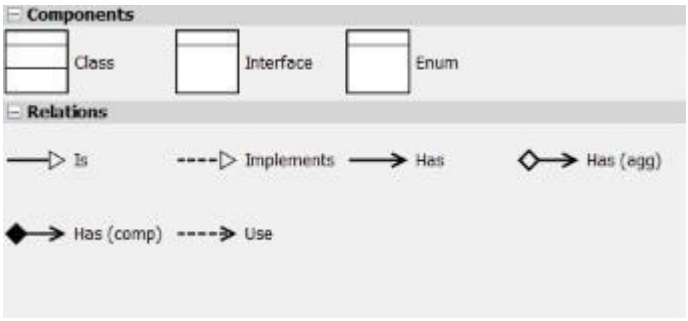
Solución: Utilizando los principios de diseño de software, modificando la lógica del juego y resolviendo los problemas de fragilidad y rigidez encontrados en el código base a partir de que los cambios propuestos modificarían clases que su funcionamiento repercutiría en otras clases adicionales a la principal, fue posible agregar más pelotas de las ya existentes.

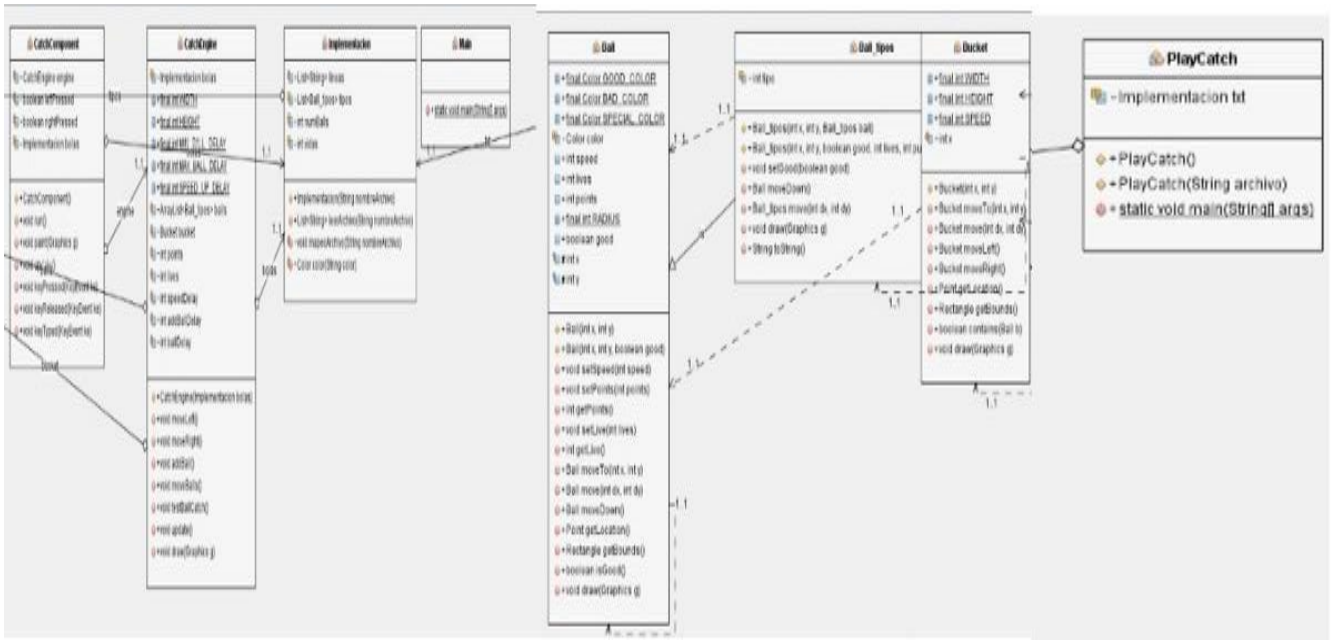
A continuación, se presenta una tabla comparativa que refleja las clases del código base vs las propuestas con la mejora implementada en el nuevo código.

Código Base	***Código Modificado***
PlayCatch.java	PlayCatch.java
CatchComponent.java	CatchComponent.java
CatchEngine.java	CatchEngine.java
Bucket.java	Bucket.java
Ball.java	Ball.java
	Implementacion.java
	Main.java
	Ball_tipos.java

Diagrama de clases:

-Contexto:





Conclusión: Tomando los siguientes datos recolectados a partir de los cálculos de las métricas de diseño orientadas a objeto, se obtienen los siguientes resultados:

Tree	A	AC	C	D	EC	I	NCP	NIP	LCC	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5
JavaProject8	0.0	0	0.0	0.71	0	0.0	9	0	0.15	10	7	2	2	0.55
(default package)	0.0	0	0.0	0.71	0	0.0	9	0	0.15	10	7	2	2	0.55
Ball			no						0.34	62	33	4	4	0.87
Ball_tipos			no						0.0	21	21	7	7	1.17
Bucket			no						0.0	0	0	0	0	0.0
CatchComponent			no						0.0	0	0	0	0	0.0
CatchEngine			no						1.0	2	0	1	1	0.89
Implementacion			no						0.0	6	6	4	4	1.0
Main			no						0.0	0	0	1	1	0.0
PlayCatch			no						0.0	3	3	3	3	1.0
DrawPane			no						0.0	1	1	2	2	0.0

Imagen 1.

Tree	NOC	NOF	NOM	NOSF	NOSM	NTM	TCC	WMC	NBD	NOP	VG	LOC	LOCm
JavaProject8	1	39	37	12	37	1	0.13	71	1	36	1	655	16
(default package)	1	39	37	12	2	1	0.13	71	1	36	1	655	16
Ball	1	11	14	4	0	0	0.32	15	0	13	1	75	2
Ball_tipos	0	1	7	0	0	0	0.0	7	1	14	1	44	0
Bucket	0	5	0	3	0	0	0.0	0	0	0	0	55	0
CatchComponent	0	4	0	0	0	0	0.0	0	0	0	0	73	0
CatchEngine	0	13	6	5	0	1	0.87	16	2	1	2	150	7
Implementacion	0	4	4	0	0	0	0.0	22	2	4	5	131	3
Main	0	0	1	0	1	0	0.0	2	2	1	2	12	1
PlayCatch	0	1	3	0	1	0	0.0	5	1	2	1	79	3
DrawPane	0	0	2	0	0	0	0.0	4	2	1	2	39	2

Imagen 2.

*1.-La clase con mayor cantidad de líneas de código (LOC) es: CatchEngine, por lo tanto, es la que tiene mayor probabilidad de tener que ser mantenida. *Véase Imagen 2.*

2.-La clase con mayor cohesión entre métodos (LCOM) es: CatchEngine por lo tanto, es la que tiene menos probabilidades de tener que ser modificada.
**Véase Imagen 1.*

3.-La clase con mayor cantidad de hijos es (NOC): Ball, por lo tanto, es la que más reutiliza código.
**Véase Imagen 2.*

4.-La clase con mayor cantidad de métodos es (NOM): CatchEngine, por lo tanto, es la que realiza más acciones de todas las clases, y posiblemente la más importante.
**Véase Imagen 2.*

Conclusión: El desarrollo y diseño de aplicaciones de la forma Model-View-Controller, resulta una forma interesante y a la vez eficiente de plantear soluciones a diferentes requerimientos planteados debido a que se presenta como un modelo simple, no costoso y flexible frente a los diferentes cambios que se le puedan hacer en un futuro.