Inicialmente se planteó una idea clara de un esquema primitivo para escribir el código de esta asignación, pero los retos comenzaron a presentarse conforme se avanzaba en el código. Algunos de estos problemas parecían enormes, pero se esfumaban tras una simple rebuscada en las notas o en las diapositivas; otros parecían un pequeño error de lógica, pero culminaban únicamente tras replantear un método por completo.

En la escritura de la clase *Robot* no hubo complicación ni con la declaración de los atributos de la clase, ni los setters, ni los getters y tampoco con el método *estaVivo. E*l primer obstáculo se presentó a la hora de digitar el método *atacar,* ya que se desconocía el cómo manejar el set y el get más allá de las declaraciones posteriores a las de los atributos privados de la clase. Tras consultar con el docente, el uso de estas palabras reservadas quedó un poco más claro, al menos para continuar el código de forma efectiva.

A la hora de plantear la clase *JuegoBatalla*, la historia fue otra. Todo transcurrió tranquilamente con los atributos, pero al no tener una idea tangible para diseñar los métodos, se decidió continuar con la instancia de los robots y su implementación al arreglo. En este punto las ideas se plasmaban con mayor fluides, sin embargo, tras utilizar un *Try Cacht,* compilar y correr la simulación, se notó que esta herramienta solo atrapaba ciertas excepciones, faltaba que el programa solo aceptara enteros dentro de los rangos establecidos y, además, la simulación se detenía cuando se encontraba algun dato no válido. Parecía un trabajo colosal y la preocupación no tardó en llegar, pero solo se requirieron únicamente unos cuantos whiles, if y else para reparar el error.

Por último, *mostrarGanador()* no dio problema alguno, la verdadera cruzada comenzó con *iniciarBatalla(),* el bloque de código restante la tarea. Luego de presentar una primera ofrenda a *javac*, los errores de sintaxis no dieron tregua por un buen rato. Problemas con declaraciones de variables llevaron a la desesperación de plantear un código completamente estático, decisión que marcaría el inicio del camino a la locura. El único paso atrás que se dio fue cuando el compilador solicitó quitar el *static* del método constructor. El código no imprimía nada después de instanciar el arreglo, se replanteó la función al menos una docena de veces y la misma seguía sin operar.

Después de una abrumante madrugada y estar a punto de ceder ante la frustración, una pequeña esperanza surgió luego de que se iluminara la habitación metafórica y literalmente, ya que estaba amaneciendo. Trasladar una línea *System.out.println()* del método problemático a *atacar,* permitió, al fin, observar lo que estaba pasando, pero la felicidad duró muy poco. El

arreglo no se estaba siendo recorrido correctamente, solo se tomaba el último objeto de este y se ejecutaba el código hasta que el robot se eliminaba a sí mismo. Posterior a entrar en un bucle con 3 o 4 propuestas de código durante toda la noche, la paz vino al fin cuando la eliminación del *static* en los atributos declarados en la primera clase resolvió todo el problema y así, la simulación corrió sin problemas.

Con esta tarea se vive un poco el ejercicio de la programación y resulta imposible no encontrar un gran parecido entre este y las matemáticas. Ambos resultan ser un parque de diversiones, con montañas rusas que conducen a la desesperación y a la esperanza de una forma casi aleatoria, con carruseles de frustración que no se tienen y con torres gigantes que provocan nauseas por estrés. Sin embargo, quienes disfrutan de la adrenalina y de los retos, placenteramente vuelven una y otra vez.

Repositorio de Git Hub:

https://github.com/vargas5320/CI-0112