

**UNIVERSIDAD DE SANTIAGO DE CHILE**

**FACULTAD DE INGENIERÍA**

**DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA**

|  |
| --- |
|  |
| **LABORATORIO 2 (PARADIGMA ORIENTADO A OBJETOS)**  **PARADIGMAS DE PROGRAMACIÓN** |
|  |

**César Antonio Salazar Silva**

|  |  |
| --- | --- |
| Profesores: | Víctor Flores |
|  | Daniel Gacitúa |
|  | Roberto González |
| Ayudantes: | Giovanni Benussi |
|  | Mauricio Rojas |
|  | Esteban Contardo |
| Fecha de Entrega: | 14 -04-2019 |

|  |  |  |
| --- | --- | --- |
|  | Santiago de Chile  1 – 2019 |  |
|  |  |  |

Tabla de Contenidos

[2. Índice de Figuras 3](#_Toc11444234)

[3. Índice de Tablas 3](#_Toc11444235)

[INTRODUCCIÓN 4](#_Toc11444236)

[1.1. Antecedentes y Motivación 4](#_Toc11444237)

[1.2. Descripción del problema 4](#_Toc11444238)

[1.3. Solución propuesta 4](#_Toc11444239)

[1.4. Objetivos del proyecto 4](#_Toc11444240)

[1.4.1 Objetivo general 4](#_Toc11444241)

[1.4.2 Objetivos específicos 4](#_Toc11444242)

[1.5 Metodologías y herramientas utilizadas 5](#_Toc11444243)

[CAPITULO 1. FUNDAMENTOS TEÓRICOS 5](#_Toc11444244)

[CAPITULO 2. DESARROLLO DE LA SOLUCIÓN 6](#_Toc11444245)

[3.1 Análisis de la solución 6](#_Toc11444246)

[3.2 Diseño 7](#_Toc11444247)

[3.3 Construcción 9](#_Toc11444248)

[3.3.1 TDA Nave 9](#_Toc11444249)

[3.3.2 TDA Asteroide 10](#_Toc11444250)

[3.3.3 TDA Disparo 10](#_Toc11444251)

[3.3.4 TDA Space 10](#_Toc11444252)

[CAPITULO 3. Resultados Obtenidos 10](#_Toc11444253)

[4. CAPITULO : Conclusiones 10](#_Toc11444254)

[5. BIBLIOGRAFÍA 11](#_Toc11444255)

Índice de Figuras

[2. Índice de Figuras 3](#_Toc11444168)

[3. Índice de Tablas 3](#_Toc11444169)

[INTRODUCCIÓN 4](#_Toc11444170)

[1.1. Antecedentes y Motivación 4](#_Toc11444171)

[1.2. Descripción del problema 4](#_Toc11444172)

[1.3. Solución propuesta 4](#_Toc11444173)

[1.4. Objetivos del proyecto 4](#_Toc11444174)

[1.4.1 Objetivo general 4](#_Toc11444175)

[1.4.2 Objetivos específicos 4](#_Toc11444176)

[1.5 Metodologías y herramientas utilizadas 5](#_Toc11444177)

[CAPITULO 1. FUNDAMENTOS TEÓRICOS 5](#_Toc11444178)

[CAPITULO 2. DESARROLLO DE LA SOLUCIÓN 6](#_Toc11444179)

[3.1 Análisis de la solución 6](#_Toc11444180)

[3.2 Diseño 7](#_Toc11444181)

[3.3 Construcción 9](#_Toc11444182)

[3.3.1 TDA Nave 9](#_Toc11444183)

[3.3.2 TDA Asteroide 10](#_Toc11444184)

[3.3.3 TDA Disparo 10](#_Toc11444185)

[3.3.4 TDA Space 10](#_Toc11444186)

[CAPITULO 3. Resultados Obtenidos 10](#_Toc11444187)

[4. CAPITULO : Conclusiones 10](#_Toc11444188)

[5. BIBLIOGRAFÍA 11](#_Toc11444189)

7

Figura 2. Diseño colisión Asteroides 8

Índice de Tablas

Tabla 1: Resultados Obtenidos. 10

# INTRODUCCIÓN

## Antecedentes y Motivación

De manera introductoria, nos remontamos a los años 80, Atari estrena lo que más adelante se conocería como un éxito, Asteroids un juego donde se debe destruir cuanto asteroide se ponga frente a la nave que se maneja en una pantalla bidimensional.

Realizar este famoso juego, es un desafío que se nos ha otorgado por la asignatura de Paradigmas de Programación. Donde este segundo de cuatro laboratorios será bajo el paradigma de programación Orientado a Objetos, utilizando Java, lenguaje que funciona bajo este paradigma y que utilizaremos para resolver este laboratorio.

## Descripción del problema

En este laboratorio se nos solicita implementar el juego Asteroids bajo el lenguaje de programación Java. Para este laboratorio, se debe implementar el videojuego descrito bajo el paradigma de programación Orientado a Objetos. Con ello se espera que podamos generar un código capaz de validar tanto como posiciones, estados de juego, colisiones entre otros aspectos a tener en cuenta para la realización de forma correcta de una partida autónoma.

## Solución propuesta

Para este apartado cabe recalcar que se nos ha otorgado una plantilla con requisitos mínimos, sirviendo estos de guía para poder hacer una buena abstracción del problema, gracias a esta es que pude identificar de manera más fácil los problemas a considerar para la representación del problema.

Los cuales son:

* Clases y estructuras que forman el programa.
* createSpace.
* moveElement.
* shootElement.
* gameLoop.
* *textifySpace y printSpace.*

## Objetivos del proyecto

# 1.4.1 Objetivo general

Plasmar a través de la práctica, los conocimientos adquiridos en catedra sobre el paradigma Orientado a Objetos.

# 1.4.2 Objetivos específicos

Analizar, comprender y manejar correctamente lo que es un Tipo de Dato Abstracto.

Desarrollar habilidades de resolución de problemas, quitando iteradores y utilizando solamente la recursión.

Poner en práctica los conocimientos del paradigma Orientado a Objetos vistos en Catedra. Como lo son clases abstractas, Interfases, objetos instanciados, herencia, acoplamiento, cohesión y un montón de conceptos que se generan en un paradigma orientado a Objetos.

# 1.5 Metodologías y herramientas utilizadas

Para hacer mas sencilla la representación se usarán datos continuos que posteriormente se discretizaran por lo que las velocidades y angulo asignadas a un objeto deben estar resultar en un desplazamiento de unidades, de no ser así puede suceder que el objeto dado el conjunto de angulo y velocidad no resulte en el movimiento de una unidad y el objeto permanezca quieto por no tener “La velocidad mínima para moverse una unidad”, en compañía de esto que es la representación bidimensional se hace constancia de los conocimientos de Física 1 y 2 adquiridos previamente para el desarrollo de la mecánica del juego como lo son Colisiones y desplazamiento, Angulo y componentes seno y coseno.

# CAPITULO 1. FUNDAMENTOS TEÓRICOS

La programación Orientada a Objetos nace en los años 60’s en el Norwegian Computing Center, se introdujo por primera vez los términos de clase, objetos y herencia.

El paradigma Orientado a Objetos es un paradigma que define su programación en términos de comunidades de objetos. Estos con características comunes se agrupan en clases, siendo estas (las clases) las plantillas para un objeto y el objeto en sí mismo es la encarnación de una clase. Los objetos son una instancia de Clase por lo que contienen datos (Atributos), tienen comportamientos y se comunican entre sí para realizar tareas (Métodos).

Dentro de las ventajas que nos brinda la programación orientada a Objetos es:

Una buena abstracción de clases, objetos y atributos nos brinda una implementación más detallada y puntual.

Nos permite reutilizar código cuando las clases se han diseñado correctamente, además se pueden utilizar en distintas partes del programa y en otros proyectos cuando la funcionalidad es la misma (Herencia). Lo que permite un rápido desarrollo, alta calidad de código, bajo costo en fase de desarrollo.

Por otro lado, otorga la facilidad de añadir o suprimir nuevos objetos esto nos permite modificar de forma más sencilla.

Idealmente tener un bajo acoplamiento y alta cohesión, las clases son independientes entre si y hacen las tareas especificas para la que fueron diseñadas.

De las desventajas podemos concluir los puntos referentes a tiempo que toma la curva de aprendizaje, algunos conceptos requieren conocimiento al implementar, se hereda código que no se usa en la clase hija cuando se extiende de una clase padre.

En paralelo es de suma importancia tener en cuenta los principios físicos para el movimiento de una partícula en este caso como lo es:

Para movimiento:

X’ = X+ velocidad\*cos (angulo)

Y’ = Y+ velocidad\*sin (angulo)

Cabe dar cuenta que es un espacio en el que “desaparece por un lado y sale por el otro” dando la ilusión de un espacio infinito, por lo que se utiliza el modulo de M y N respectivamente para los nuevos X e Y.

Para Colisiones entre Asteroides:

Se hará un cambio de angulo y velocidad al azar con la seed entregada.

Para Colisiones Nave-Asteroides:

Se verifica si colisiona la nave con una lista de Asteroides y luego de verificar, se procede a modificar el estado de Juego (2 = Jugando, 1 = Gana, 0 = Pierde).

Para Colisiones Disparos-Asteroides:

Se Verifica la colisión entre una lista de asteroides y una lista de Disparos, se elimina el disparo colisionado y el asteroide si su radio es mayor a el mínimo, se divide en P particiones.

Para Verificar Colisiones:

Para identificarlas se usará el siguiente modelo:

Tomando todos los objetos con un radio propio, es decir:

Distancia entre Ambos objetos es menor o igual a la Suma de sus radios.

De cumplirse lo anterior los objetos están interceptados entre sí.

# CAPITULO 2. DESARROLLO DE LA SOLUCIÓN

# 3.1 Análisis de la solución

. Para tener una solución a este problema debemos preguntarnos, el cómo queremos representar nuestros objetos como lo es el Disparo, Nave, Asteroide, la relaciones entre estos, el manejo de la velocidad, cambio de dirección en cada elemento.

Para ello haremos el uso de lo llamaremos TDA, Tipo de Datos Abstractos los que se conforman de:

- Una Representación: Es como se va a representar nuestros objetos, ya sea, una lista donde la primera posición signifique la variable de posición, la segunda una lista con componentes que se vayan a utilizar, etc. Es como armaremos a conveniencia la estructura en la que vamos a trabajar En este caso será la Clase asociada al objeto a representar, tanto como para un Asteroide, un disparo o una Nave.

- Constructor: En este laboratorio estará asociado a un método dentro de la clase que queremos instanciar, que a través de este se convertirá en un objeto, lo utilizaremos para crear nuestro TDA del objeto asociado.

- Función de pertenencia: No fue utilizada en este laboratorio

- Selectores: Métodos que permiten acceder a los elementos almacenados en nuestro TDA objeto (atributos de la clase). Esto nos permite poder utilizarlos en otras funciones para ejercer operaciones o comparaciones logicas.

- Modificadores: Esta función nos permite modificar un elemento del TDA, ya sea para cambia de posición en X en nuestro caso, Angulo, velocidad, etc.…

- Funciones Varias u otras son las que operan sobre este esquema.

Dado lo anterior y bajo los parámetros del enunciado pasamos a desarrollar un prototipo de solución el cual debe contener:

- Función que crea un espacio con asteroides, nave y disparos.

- Hacer un buen uso del TDA con cada función antes mostrada y definirle de forma clara y precisa.

- Hacer una función que “mueva la nave” considerando un sistema de ángulos fijos. Además, verificar si la nave ganó, perdió o sigue en juego.

- Función que genere un disparo y además mueva el tablero en t+1. Así mismo verifique el estado de juego.

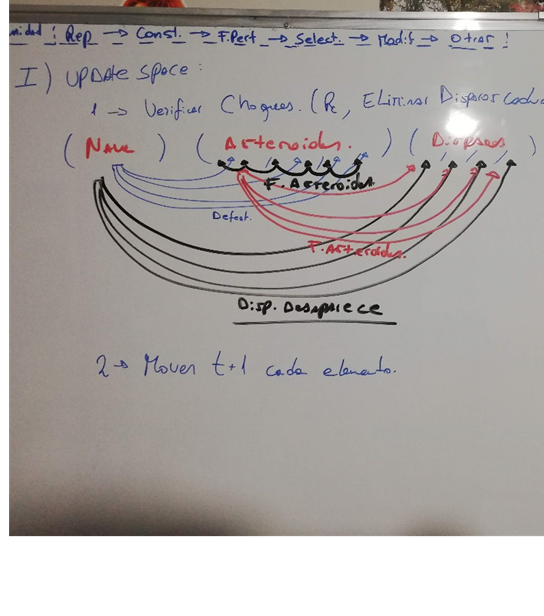
- LoopGame método que actualiza el juego solamente en un t+1.

- Y un método que pase del TDA Space a un String legible y otra que imprima este String por pantalla.

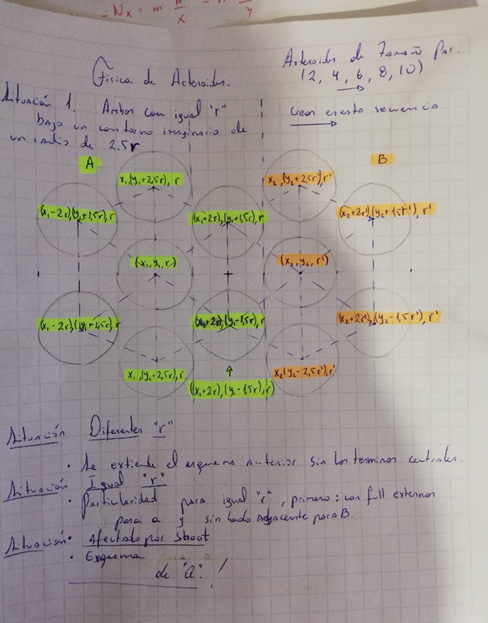
También recordando que nuestro desarrollo debe respetar el paradigma Orientado a Objetos, con sus respectivas Clases y sus instancias, respetar el acoplamiento y cohesión. Para generar un buen código y reutilizable

# 3.2 Diseño

Como se ilustran en mis bosquejos a continuación, debemos además de mover un objeto en un tiempo t+1, se debe verificar las colisiones entre Nave y asteroides, también entre asteroides y disparos, y una física de asteroides que divida un asteroide chocado en trozos más pequeños con una lógica coherente a la física real:

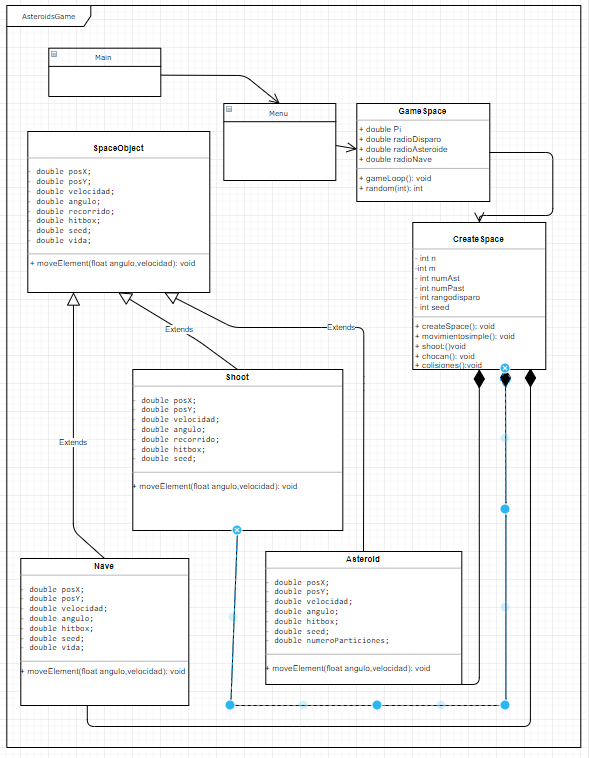


*Figura 1. Diseño interacciones entre TDAS*

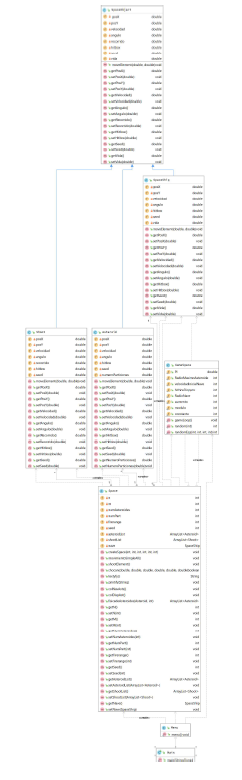


*Figura 2. Diseño colisión Asteroides*

El programa tenía en la fase de diseño en un diagrama UML de la forma :



El diagrama final viene adjunto en un PDF llamado Uml el cual contiene la forma final del proyecto con todo detallado, de igual forma mostraré una miniatura de este a continuación. (Como se puede apreciar el diagrama es muy grande, es por ello que viene adjunto)



# 3.3 Construcción

# 3.3.1 TDA Nave

La representación de la Nave está dada por la clase SpaceShip, esta es la plantilla para un Objeto Nave, donde se tiene que en Variables esta la información de la posición X, luego posición Y, su respectiva velocidad, seed traspasada, estado de juego que puede tomar 0 para Defeat, 2 para Win y 1 para Playing, Por último, el angulo en el que se desplaza la Nave como atributos de la misma y es clase hija de ObjectSpace que representa a un Objeto espacial.

# 3.3.2 TDA Asteroide

La representación de un proyectil está dada por la clase Asteroid que hereda de SpaceObject, clase que contiene el comportamiento general de cualquier objeto espacial como un disparo, nave o Asteroide. Donde se tiene que en Variables esta la información de la posición X, luego posición Y, su respectiva velocidad, seed traspasada, el angulo en el que se desplaza el Asteroide y su Radio, que es el hit box con el cual interactúa con los demás objetos y un método de movimiento simple.

# 3.3.3 TDA Disparo

La representación de un disparo está dada por la clase Shoot, que hereda de SpaceObject , clase que contiene el comportamiento de cualquier objeto que interactúa en Space. Donde se tiene que en Variables esta la posición X, luego posición Y, su respectivo angulo y velocidad, su largo que es el recorrido total del disparo y la seed propia del disparo como atributos de la clase y un método de movimiento simple.

# 3.3.4 TDA Space

La representación del espacio está dada por la Clase Space, esta contiene como atributos los valores N, M, numero de Asteroides, numero de Particiones, Alcance de Disparo, lista de Asteroides, disparos y el elemento nave.

En sus métodos podemos ver que contienen las interacciones del Space como lo son colisiones, movimientos de todos los objetos en un tiempo t+1 y convertir un Space a String y poder mostrarlo al usuario

# CAPITULO 3. Resultados Obtenidos

* Clases y estructuras que forman el programa.
* createSpace.
* moveElement.
* shootElement.
* gameLoop.
* *textifySpace y printSpace.*

El laboratorio consta de los siguientes Requisitos Obligatorios, descontando el Buen uso del TDA queda:

|  |  |  |
| --- | --- | --- |
| **Función** | **Descripción** | **Resultado** |
| createSpace | Crear espacios validos de forma aleatorea | Logrado |
| moveElement. | Mover elementos y verificar física de juego, aplicable a todos, pero con énfasis en la nave. | Logrado |
| shootElement | Disparar un elemento y verificar física de juego para un tiempo t +1 | Logrado |
| gameLoop | Generar un bucle de juego, lógico físicamente. | Logrado |
| *textifySpace y printSpace.* | Convertir un Space en texto e imprimirlo en pantalla | Logrado |
| Menu | Menu para realizar una interacción con el usuario | Logrado |

# CAPITULO : Conclusiones

Este es un laboratorio que me produce satisfacción entregar, dado que el esfuerzo puesto en este código, dio los resultados que yo esperaba, las colisiones se verifican correctamente, los movimientos en un tiempo t a t+1 cumplen con lo que designe para ellos en la relaciones entre los objetos, me gustaría probar casos bordes en los que mi código no funcione o haya incongruencias, pero según estuve revisando, al menos en casos normales debe funcionar. Quedo con un gusto de insatisfacción con respecto a lo explicado previamente, en cuanto a la congruencia de mi algoritmo y su funcionalidad en todos los casos posibles y verificación correcta de choques en todos los casos. Pero a lo que a mi concierne, se ejecuta de forma correcta.

En cuanto al paradigma empleado para resolver este Laboratorio, puedo decir, que aprendí de forma satisfactoria el paradigma Orientado a Objetos, me gustaría haber tenido más tiempo para dedicarle en cuanto a los comentarios, o hacer un análisis más profundo y utilizar conceptos como los es la herencia e interfases de una forma más consiente.

En paralelo, a todo esto, en mi percepción, quedo satisfecho por lo aprendido y el uso consciente de este paradigma, me ayudo a aprender de mejor manera una forma distinta de afrontar el mismo problema.

# BIBLIOGRAFÍA

Referirse a “Paradigmas de Programación POO”, Pagina web con contenido de ayuda para el paradigma Orientado a Objetos:

* URL: <https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/index.html>

Referirse a “Moodle, Plataforma en la cual se aloja el material del curso”:

* URL: <http://www.udesantiagovirtual.cl/moodle2/course/view.php?id=7167>