Utilizar el POO con Pharo SmallTalk en el desarrollo de juegos

[a]Agustin Intile; [b]Rosalia Sánchez; [c]Lourdes Vega; [d]Victoria Velazquez;

[a] Universidad Tecnológica Nacional, Curso 2.1, [a]intileagustin@gmail.com
[b]rosancheza2@gmail.com
[c]lourdesvega2002@gmail.com
[d]vicvelazquez002019@gmail.com

Resumen

En este trabajo, demostraremos cómo el Paradigma Orientado a Objetos facilita la representación de los elementos, y como la flexibilidad de SmallTalk, nos permite ajustar y perfeccionar nuestro programa a medida que avanzamos en el desarrollo

En este proyecto, reflejamos una visión detallada de nuestra experiencia al aplicar este paradigma, en la implementación del juego "Sumas Anticipadas", destacando tanto los aspectos conceptuales como las soluciones prácticas encontradas durante el proceso.

1. Introducción

Pharo es un entorno de desarrollo y un lenguaje de programación basado en Smalltalk, conocido por su simplicidad, claridad y potencia. Smalltalk ha influido en el diseño de muchos otros lenguajes de programación orientados a objetos, y sus principios han sido adoptados en diversas formas por otros entornos de desarrollo.

¿Cuál es nuestro objetivo? Nuestra misión es demostrar cómo Pharo se puede utilizar para la comprensión del POO mediante la ejecución de un juego.

El juego que vamos a demostrar a lo largo de este trabajo se denomina “Sumas Anticipadas”. El proyecto se desarrolla en un tablero cuadrado de orden 2n+1, y la partida continúa hasta que todos los números estén tachados, o se vuelva imposible continuar. El ganador, es el jugador con el puntaje más alto al final de la partida.

El desafío estratégico del juego consiste en analizar la elección de números para maximizar el puntaje propio mientras obstaculiza las opciones del oponente. La anticipación y la planificación de movimientos, son fundamentales para alcanzar el éxito en "Sumas Anticipadas".

2. Desarrollo

Como punto de partida, se quiere que el futuro jugador tenga a su alcance múltiples opciones dentro del juego. Por ello, implementamos las siguientes elecciones: Iniciar partida, Explicar modo de juego, Cerrar juego.

Fig 1: Interfaz de Inicio. Opciones de juego.

El juego de “Sumas Anticipadas” utiliza un tablero con números random y dos jugadores podrán hacer disposición del mismo. El primer jugador, podrá seleccionar filas mientras que el segundo jugador elige columnas.
Una vez inicializada la partida, los jugadores podrán colocar su nombre identificativo y seleccionar la dificultad. El tablero donde se desarrolla el juego es *dinámico*, es decir, los números en las casillas cambian en cada partida.

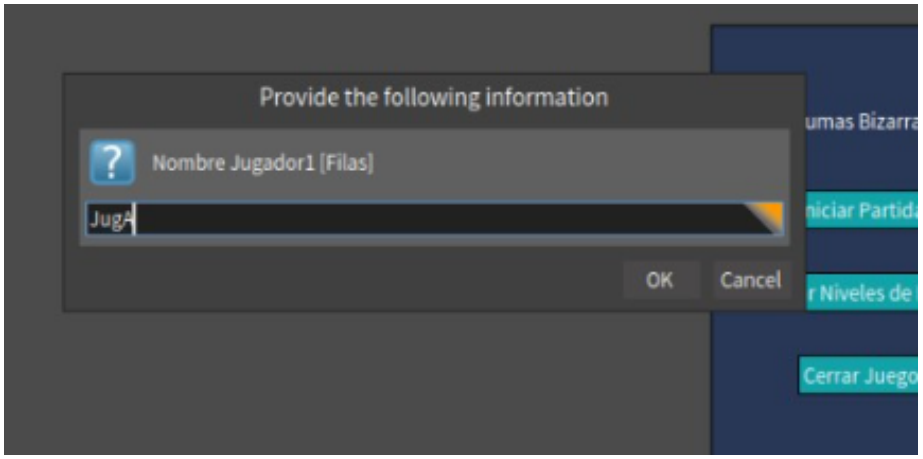

Fig 2: Ingresar nombre de los jugadores.

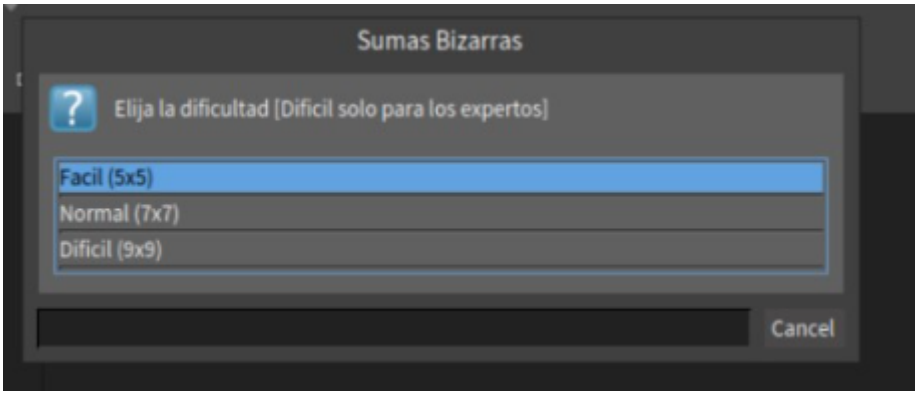

Fig 3: Selección de dificultad.

Cuando la competición esté en marcha, tendrán una visualización del tablero de juego. Allí, tendrán múltiples opciones como: pasar turno, reiniciar, cerrar. En caso de que, un jugador, desee seleccionar una casilla que no le corresponde o un bloque ya utilizado previamente, el juego generará una notificación detallando el inconveniente. También se emitirá una alerta indicando el ganador del encuentro.

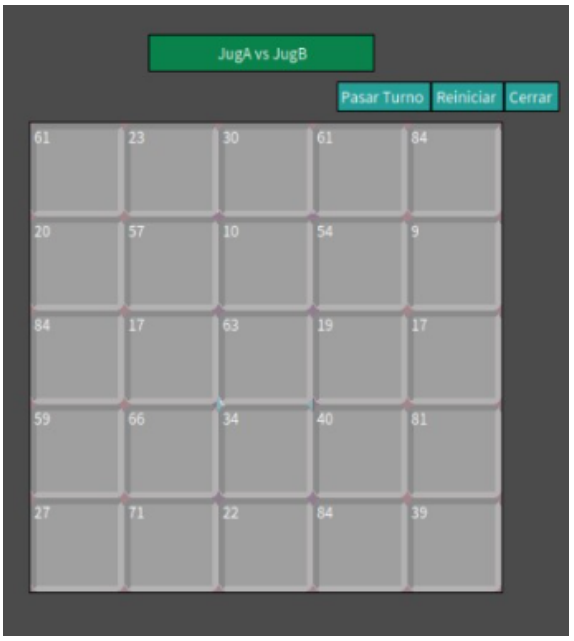

Fig 4: Tablero inicial.

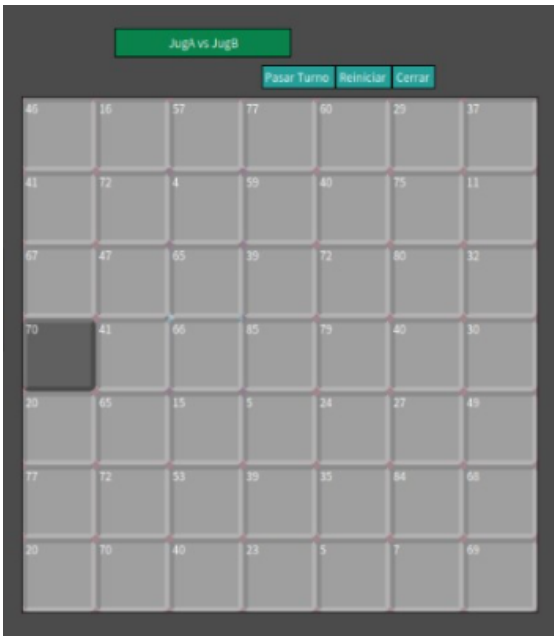

Fig 5: Primer jugada realizada en el tablero

Como se puede observar en la figura 5, las casillas seleccionadas o ya jugadas, se irán marcando en un tono gris oscuro. Lo que permitirá a los jugadores identificar fácilmente las casillas disponibles. Así sucesivamente, hasta finalizar el juego. Esta modalidad de juego también se aplica al tablero 7x7 (Modo Normal).

Modo Difícil: Sumas Bizarras

Llegado a este punto, ¿Por qué no implementar una característica extra? Aplicar algún modo de juego o componente que permita desafiar a los jugadores y mejorar su entretenimiento. Con el propósito de evaluar nuestras destrezas y habilidades como desarrolladores, se optó por incorporar un elemento adicional al software.

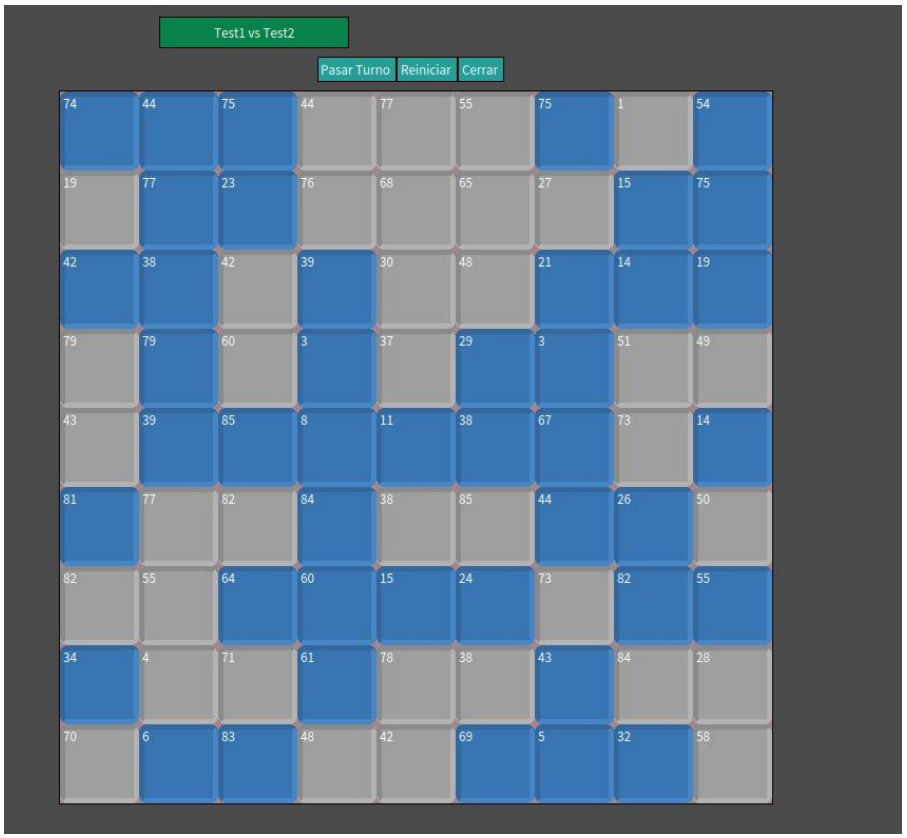

Fig 6: Modo Difícil tablero.

Este es el modo *"Sumas Bizarras"*, que además de incorporar una matriz 9x9 como tablero de juego, como se observa en la figura 6, incorpora nuevos factores que los jugadores deberán tener en cuenta durante la partida.

Los *bloques bizarros* constan de dos personalidades: buena y enojada. Cada una de ellas, brindarán un bonus o una penalización en el puntaje total del jugador que seleccionó dicho bloque.

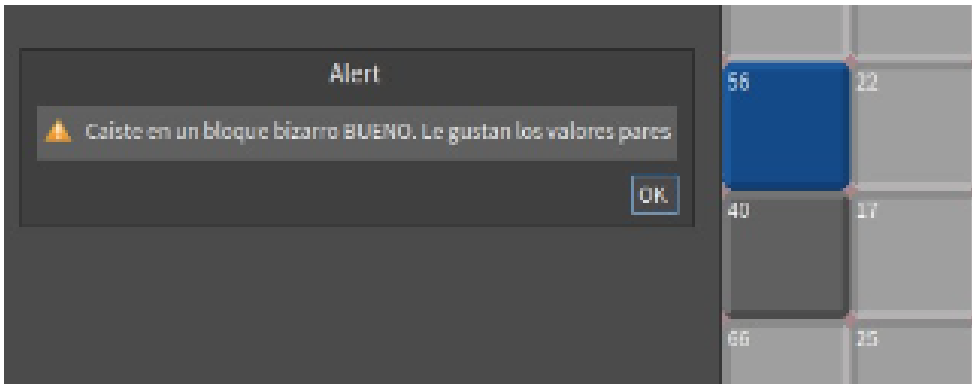

Fig 7: Bloque bueno.

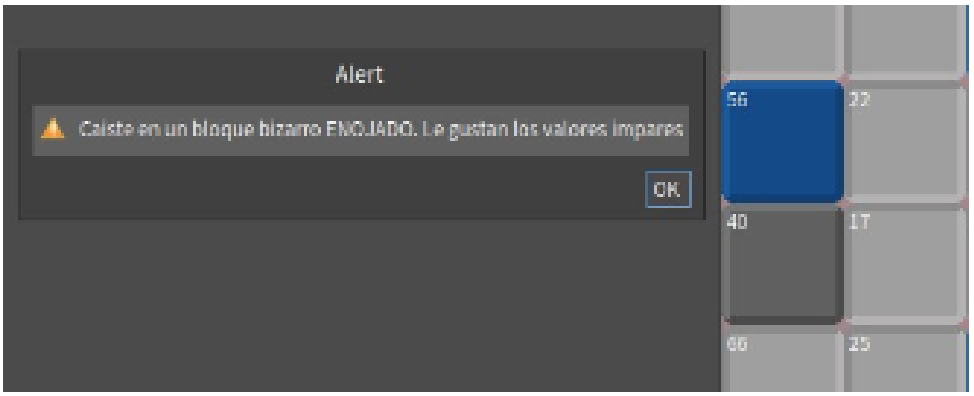

Fig 8: Bloque enojado.

La ubicación de los bloques bizarros es independiente del número de la casilla, se ubican de manera aleatoria alrededor del tablero. Por ende, en cada partida, estarán en posiciones distintas.

Clases

Clase "Inicio"

Clase creada para abrir el menú de opciones del juego mediante el código *"Inicio new"* , subclase de la clase *"BorderedMorph"*, cuenta con los botones *"Iniciar Partida" , "Ayuda", "Cerrar"*.

En la clase *"Inicio"* se encuentran los siguientes métodos:

- *"Ayuda":* Método que crea un cuadro de texto que explica al jugador las bases de la partida (La dificultad y el cómo está afecta al tablero.
- *"initialize":* Método que inicializa la parte gráfica del menú, creando la instancia de *"BorderedMorph"* y las instancias de botones con *"boton: ejecutar:"*
- *"boton:ejecutar:":* Método auxiliar que crea las instancias de *"SimpleButtonMorph"* y les asigna el bloque de código a ejecutar.
- *"reiniciar":* Método auxiliar de *Ayuda* que vuelve a crear una instancia de *Inicio*.
- *"terminar":* Método que elimina con *"self delete"* a la instancia de *Inicio*
- *"inicioPartida":* Método que crea una instancia de la partida con *"Partida new openInWorld"*, elimina al menú y comienza el juego.

Clase "Bloque"

Los bloques son instancias de la clase *"Bloque"*. Esta es una subclase de la clase *"SimpleSwitchMorph"* , propia de Pharo, que nos permite crear objetos con cambios de estado (encendido y apagado como un Interruptor).

En la clase *"Bloque"* se utiliza el método *"initialize"* para definir los atributos gráficos del bloque (colores de apagado y encendido) y con un randomizador definir los valores de cada bloque, resguardados en "*n*".

La clase cuenta con los métodos *"mouseAction", "mouseMove"* y *"mouseUp"* los cuales permiten al Morph detectar una interacción del mouse.

Clase “Bizarro”

Subclase de *“Bloque”*. Los bloques que mutaron y se volvieron bizarros tienen un atributo nuevo *“personalidad”*, definida por un randomizador al inicializar el objeto, tiene dos valores posibles: [1,2]. Los bizarros tienen gustos, si es 1 (Bueno) le gustan los valores pares; si es 2 (Enojado) le gustan los impares. Si se cumplen estas condiciones, los Buenos sumarán más puntos y los Enojados descontarán menos puntos.

Clase “Jugador”

Los jugadores son los principales actores del juego, cuentan con un *nombre, puntaje, estado y coordenadaLock*. La progresión de la partida se representa en el estado, con los valores *“Jugando”* y *“En Espera”*, el puntaje se va a sumar o restar con los métodos personalizados *“sumarValor”* y *“restarValor”*. Para inicializar al jugador con el nombre y la coordenada de inicio existe el método *“crear: inicia:”* que recibe un nombre y un valor númerico.

Clase “Partida”

La partida contiene los atributos de *tablero, la dificultad, los jugadores y la ronda actual*. Es una subclase de *“BorderedMorph”* al igual que *Inicio*, que permite definir su tamaño, bordes, color y anexar otros Morph. En la clase “Partida” se encuentran los siguientes métodos:

- “*initialize”:* Método que inicializa al *“BorderedMorph”*, a la *“ronda”* y a los *“jugadores”* con los nombres de los jugadores y zona de inicio. Llama al método *“elegirDificultad”* que le va a dar el tamaño del Array2D al *“tablero”* y en cada posición llama al método *“crearBloque:at:” o “crearBizarro:at:”* que instancia un bloque.

- *“elegirDificultad*”: Método setter que muestra una lista de opciones para seleccionar la dificultad del juego.

- *“dificultad”:* Método getter de “*elegirDificultad”*.

- *“funcionalidades”:* Método auxiliar de construcción de la barra de control con los botones *“Cerrar”, “Reiniciar” y “Pasar Turno”*. Además cuenta con un cartel que muestra los nombres de los jugadores.

- *“boton:ejecutar:”*: El mismo método existente en la clase “*Inicio”* para crear los botones.

- *“mostrarOponentes”:* Método usado en *“funcionalidades”* para crear el cartel de los jugadores.

- *“reinicio”*: Método que pregunta al usuario si quiere reiniciar, en caso de aceptar destruye la instancia de *“Partida”* y crea una nueva.

- *“terminar”*: Método que pregunta al usuario si quiere cerrar el juego y destruye la instancia actual de *“Partida”*.

- *“pasar”:* Método que recorre el tablero en busca de una celda encendida y devuelve al jugador correspondiente su coordenada, modificando los estados de cada jugador.

- *“crearBloque:at:”:* Método que crea una nueva instancia de bloque, define su posición dentro del tablero y qué evento se ejecuta cuando se lo selecciona.

- *“crearBizarro:at:”*: Método que utiliza un randomizador para definir si crea una instancia de bloque normal o una instancia de “*Bizarro”* en el tablero.

- *“procesarEleccionAt:at:”*: Método principal que va a procesar los valores *“n”* de cada bloque y los va a sumar al puntaje del jugador correspondiente. Verifica que el bloque no esté tachado o fuera del alcance del jugador. Si existe un bloque *“Bizarro”* llama al método *“procesarBizarro: jugador:”*. Al finalizar el proceso se llama al método *“verificarResultados”*.

- *“procesarBizarro:jugador”*: Método que modifica el puntaje total del jugador según la personalidad del *“Bizarro”*, descontando o sumando una cantidad definida.

- *“verificarResultados:”*: Método que recorre todo el tablero, en caso de que exista alguna celda disponible, continúa el juego. Si el tablero está lleno llama a *“mostrarGanador”*.

- *“mostrarGanador”*: Método que compara los puntajes de ambos jugadores y muestra con un *UIManager* un mensaje que muestra el total de rondas, el puntaje del ganador y el perdedor, y el nombre del ganador. Da la opción a reiniciar o cerrar el juego.

3. Conclusión

A lo largo de este artículo, hemos abordado la implementación y desarrollo de un juego denominado "Sumas Anticipadas" en el lenguaje Pharo, centrándonos en los principios de la Programación Orientada a Objetos (POO).
Si bien el juego que debíamos realizar es simple, su ejecución en este lenguaje fue un desafío para nosotros. Debido a nuestro desconocimiento inicial, e inexperiencia sobre cómo llevar a cabo la idea, así como el tiempo limitado que disponíamos.

Luego de realizar un análisis intensivo de las reglas del juego y crear un diagrama de clases preliminar que nos sirva de guía para el desarrollo inicial del programa, pudimos comenzar la puesta en marcha del código. Gracias a estos pasos iniciales, obtuvimos un mejor panorama de cómo abordar el trabajo, sus clases, relaciones y posibles métodos. Conforme íbamos avanzando el código, también actualizamos el diagrama de clases para utilizarla de forma constante como herramienta en la conceptualización y comprensión profunda del software.

El mayor desafío de nuestro equipo, fue la implementación del lenguaje Pharo (SmallTalk) en un proyecto extenso y con muchos elementos del POO. A su vez, otra dificultad fue la utilización de Morphic. Ya que, la documentación es escasa y mayormente en inglés. Con mucho esfuerzo y sacrificio, superamos los obstáculos y logramos llevar a cabo

“Sumas Anticipadas” de manera satisfactoria. Dejándonos en evidencia, la necesidad de aprender a trabajar en equipo, lidiar con la frustración, la importancia de la disciplina, entre otras cualidades que requiere un ingeniero en sistemas de información.

4. Bibliografía

Stéphane Ducasse and Gordana Rakic with Sebastijan Kaplar and Quentin Ducasse. (2022). Pharo 9 by Example: https://books.pharo.org/pharo-by-example9/pdf/2022-03-26-index.pdf

Stéphane Ducasse. (2021). Pharo with Style: http://books.pharo.org/booklet-WithStyle/pdf/WithStyle.pdf

Dirección de Cultura y Educación, Provincia de Buenos Aires. Juegos que pueden colaborar en el trabajo en torno al cálculo mental - Área Matemática: https://progresionescaba.bue.edu.ar/fileprog/juegos_que_pueden_colaborar_en_el_trabajo_en_torno_al_calculo_mental.pdf

Jigyasa Grover.(2023).Smalltalk: https://learnxinyminutes.com/docs/smalltalk/

Philippe Marschall.(2015).Object Initialization: https://github.com/SeasideSt/Seaside/wiki/Object-Initialization

Peteruhnak.(2016). Pharo UIManager API notes: https://gist.github.com/peteruhnak/8ac6667b8eb11daee71517d0172b7355

Diagrama de clases: Sumas bizarras(2023): https://app.diagrams.net/#G1ySFLr0BefgdI3nwaWDHWRDpaVpbrDpbH