UNIVERSIDAD DE SANTIAGO DE CHILE

Informe Laboratorio #4 – Proyecto Semestral
Paradigmas de Programación

Paradigmas de Programación

Profesor: Roberto González Ibanez

Clemente Aguilar Osorio

19960801-0

Julio 2022

Introducción

El siguiente informe tiene como objetivo describir y analizar la solución propuesta para el Laboratorio #4 del Proyecto Semestral de Paradigmas de Programación. Se describirá el diseño planteado para la solución escogida, el acoplamiento necesario mediante paradigma dirigido por eventos para suplir las necesidades de esta entrega, al igual que los resultados obtenidos, concluyendo finalmente respecto al trabajo realizado.

Descripción del problema

El problema propuesto consta de implementar una réplica del popular juego de mesa Dobble. Esto considera los siguientes elementos para jugar: mazo(s) de cartas, jugadores y una instancia de juego. Cada carta contiene una cantidad fija de elementos, y cada par de cartas perteneciente a un mazo de Dobble tienen un solo elemento en común. El ciclo de juego consta en identificar el elemento en común entre las cartas mostradas. Es posible jugar distintos modos de juegos que varían en la cantidad de cartas, mazos y jugadores.

Descripción del paradigma

Conectando con el trabajo realizado en el laboratorio anterior, centrado en el paradigma orientado a objetos, en esta ocasión se implementó una solución multiparadigma. Esto significó diseñar una interfaz y control de eventos siguiendo los lineamientos del paradigma dirigido por eventos, llamando a métodos y clases de la representación del juego mediante paradigma orientado a objetos.

A diferencia de los paradigmas vistos previamente en el curso, esta representación se interesa en los sucesos que ocurran en la ejecución de un programa. Este enfoque exige alejarse de una mirada algorítmica secuencial, centrándose en responder a las acciones de un usuario o las excepciones y manejo de datos que ocurran dentro de un sistema.

Para un correcto acoplamiento de los paradigmas en la aplicación, es necesario identificar y delimitar la responsabilidad Modelo-Vista dentro de la solución a adoptar, diseñando una interfaz y control que dependa de los métodos y modelos definidos en el laboratorio anterior, pero que estas clases y estructuras sean agnósticas a la interfaz que los vaya a representar.

Análisis del problema

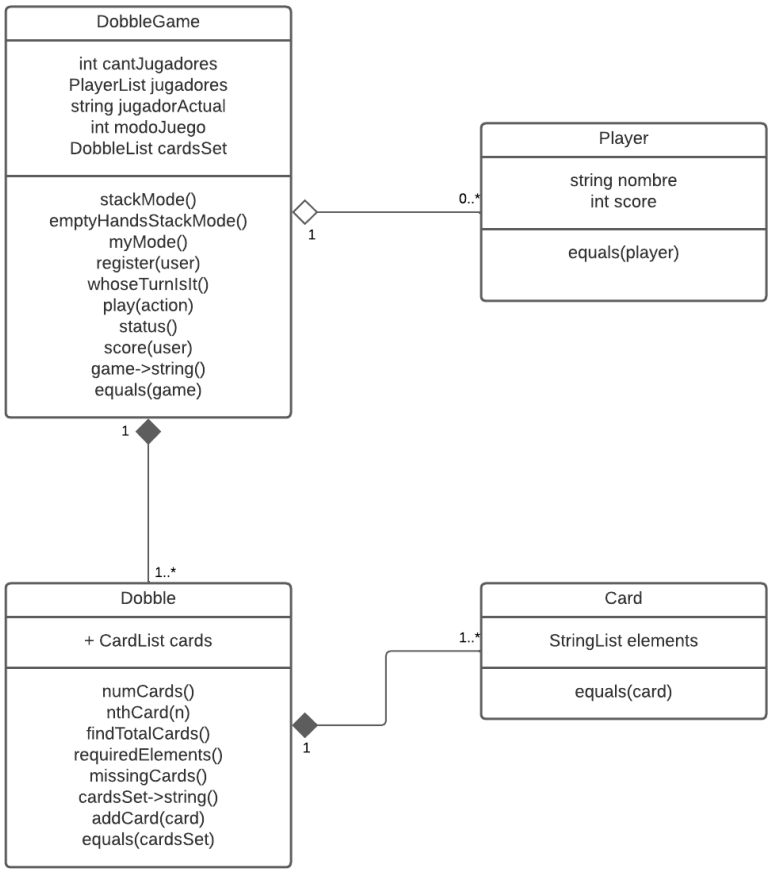
Para poder representar los elementos, acciones y pasos que componen un juego de Dobble, es necesario abstraer distintos conceptos y procesos. Para esto, se redefinirán las clases seleccionadas para la entrega del Laboratorio #3:

Inicialmente, al tratarse de un juego de cartas, se necesita un conjunto de métodos y atributos que den lugar a la estructura “Card”. Una vez definida la estructura carta, se necesita un mazo de cartas para jugar, el cual será “Dobble”. Lo último que falta para poner un juego en marcha, son jugadores y una instancia de juego.

Para esta iteración del laboratorio, además, se apunta a simular un juego de Dobble a través de la consola de forma más interactiva.

Diseño de la solución

La representación del modelo escogido comenzó desde un diagrama de análisis, señalando las clases y relaciones que regirán sobre el modelo. El diagrama generado es el adjunto a continuación:



En este diagrama se pueden apreciar 4 clases:

- DobbleGame
- Player
- Dobble
- Card

DobbleGame: Esta clase representa un juego de Dobble. Sus atributos son cantidad de jugadores, lista de jugadores dentro del juego, nombre del jugador actual, modo de Juego y una lista de mazos con los que se está jugando.

Player: Esta clase modela a cada jugador. Sus atributos son nombre del jugador y puntaje actual del jugador.

Dobble: Esta clase representa al mazo de cartas del juego Dobble. Su único atributo es una lista de cartas Dobble.

Card: Esta clase modela una carta de juego. Su único atributo es una lista de elementos que contiene la carta.

Para el manejo de los eventos se diseñó una interfaz gráfica que sea responsable de encapsular, identificar y decidir en base a posibles eventos que sucedan dentro del programa.

Aspectos de implementación

Para la implementación del diseño y la confección de la interfaz se trabajó con el ambiente de trabajo Apache NetBeans IDE 14, trabajando con el lenguaje Java, incluido en OpenJDK versión 11.

Teniendo las clases que representan la interacción dentro de un juego Dobble, se creó una clase encargada de llamar a la creación de juegos y los métodos que preceden a esta estructura. Esta clase será la responsable de comunicarse con la interfaz gráfica. Lograr esto permite que los métodos existentes dentro de clases como Dobble, Card o Player sean agnósticas a la lógica y requerimientos de la interfaz.

Durante la realización del laboratorio hubo un proceso de redefinición de las clases existentes en el diagrama de análisis, resultando en un modelo más refinado y apto para la solución que se buscó implementar. Esto corresponde al siguiente diagrama de diseño:

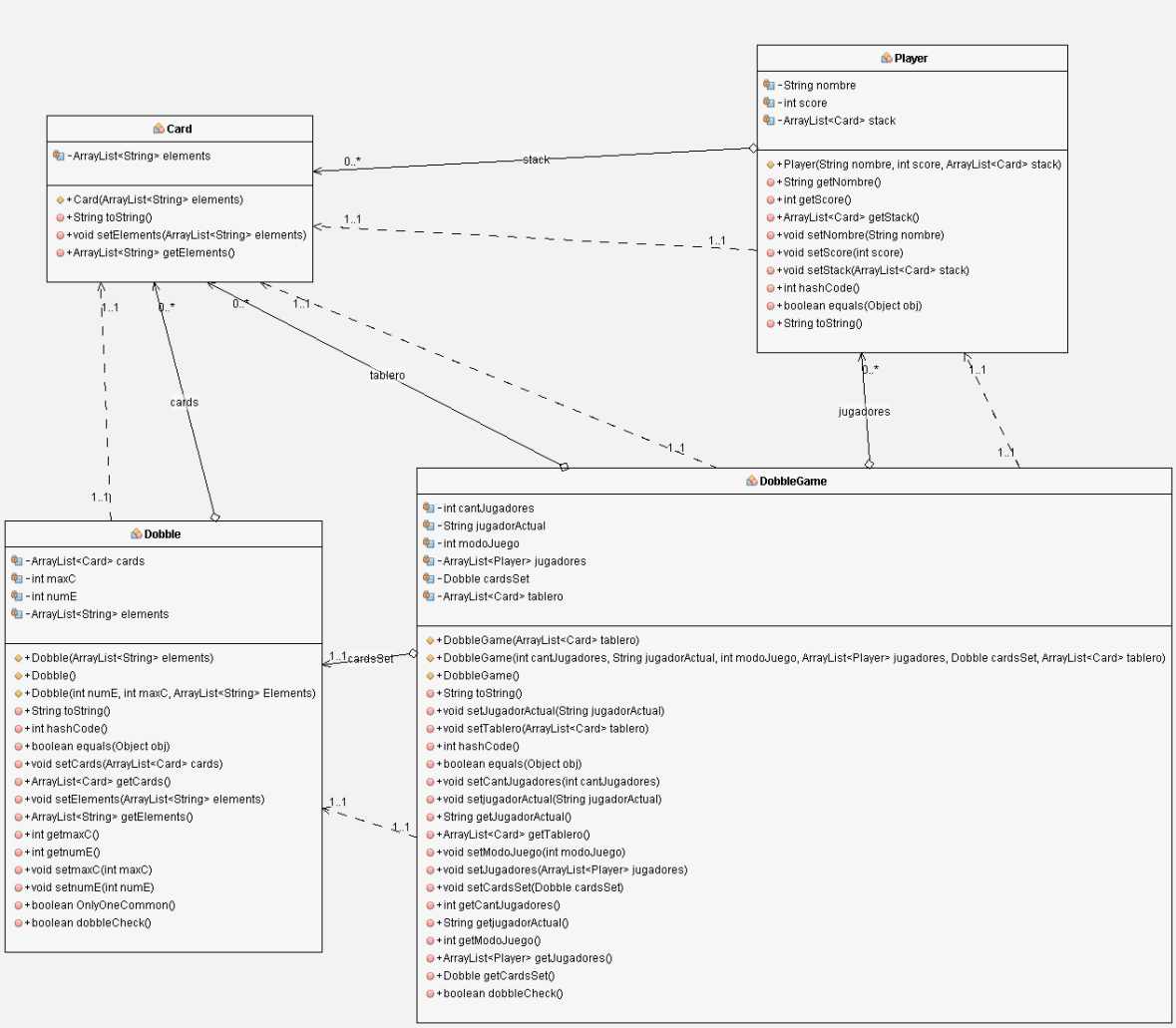


Diagrama UML para las clases Card, Dobble, Player y DobbleGame.

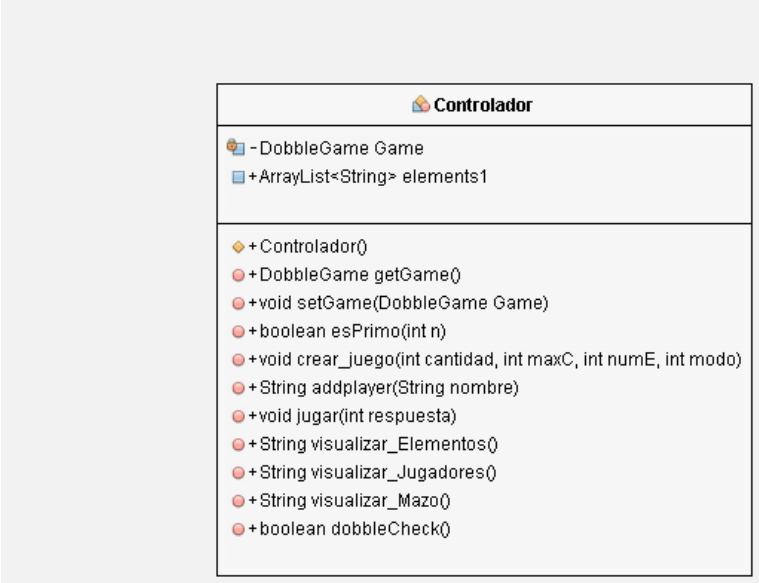
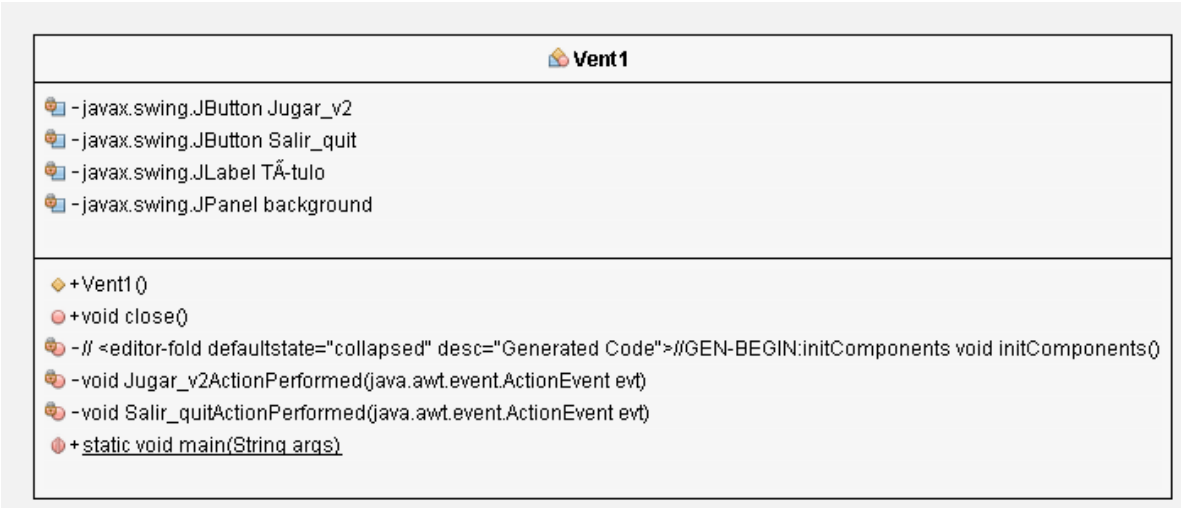
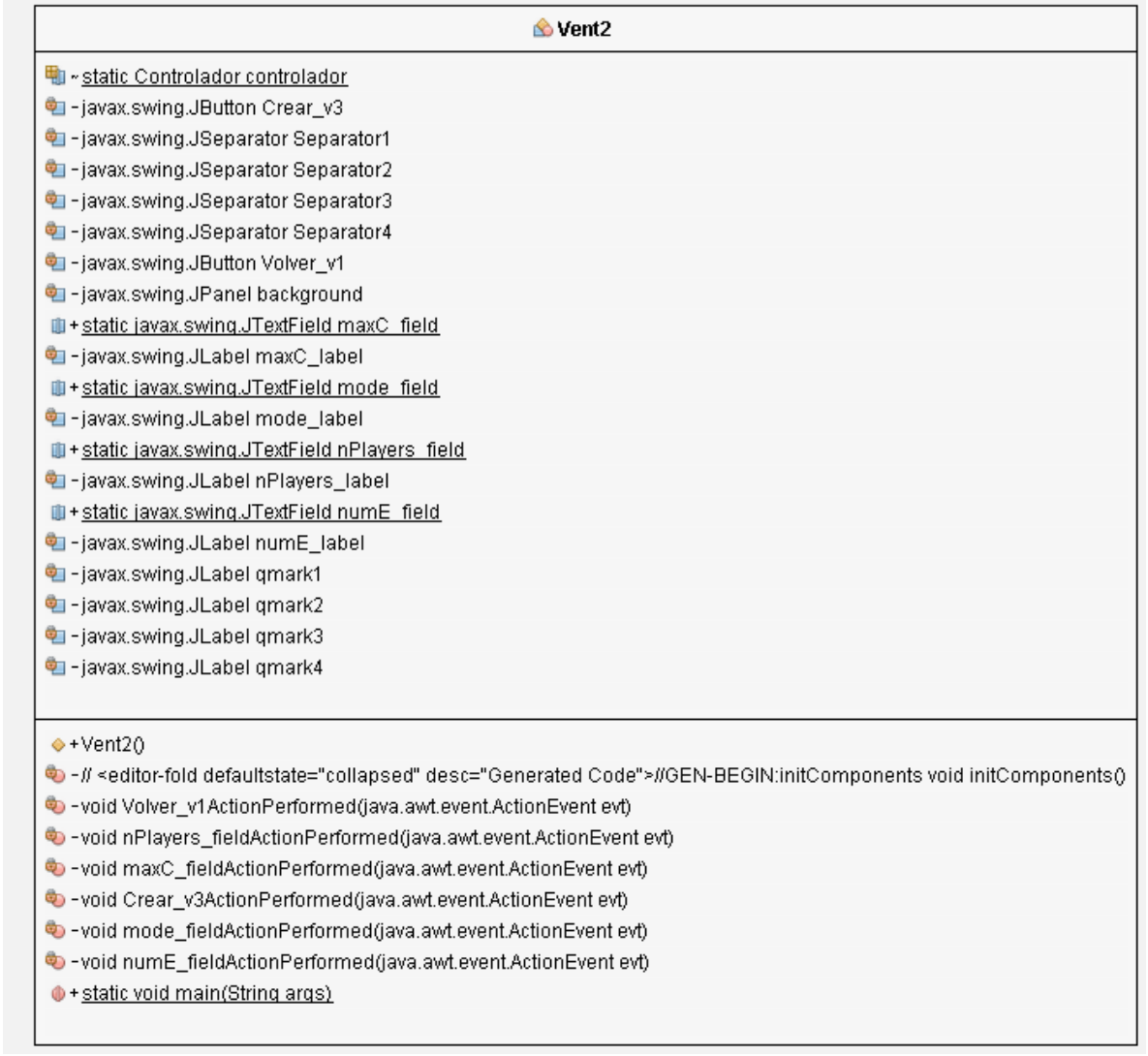
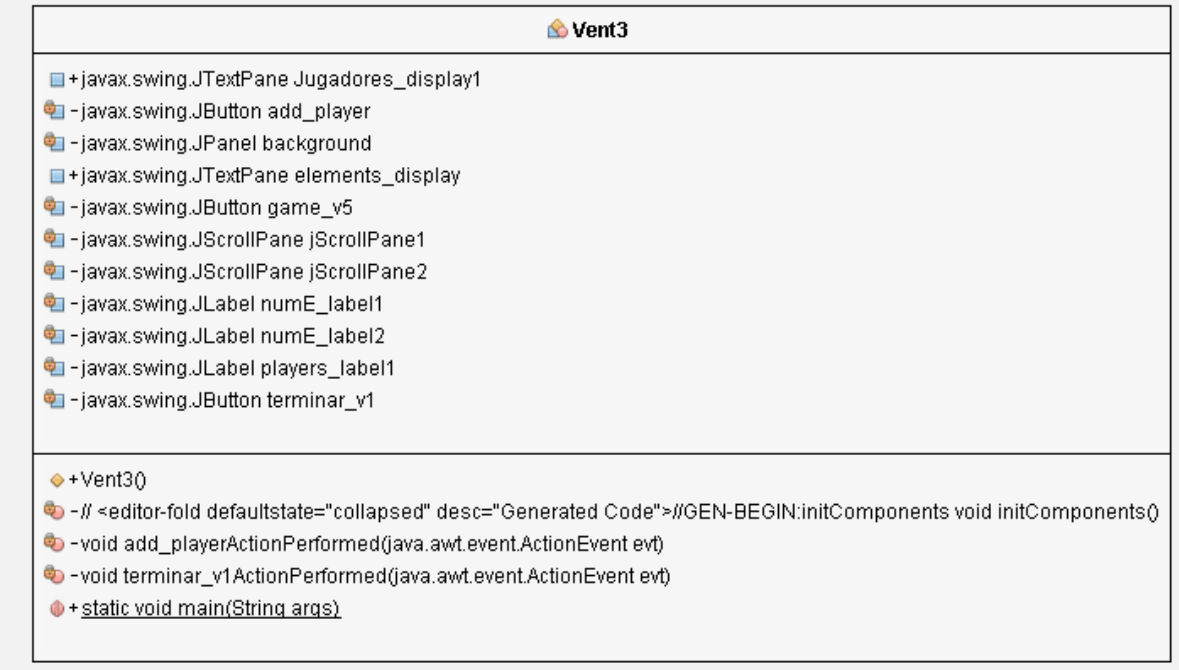


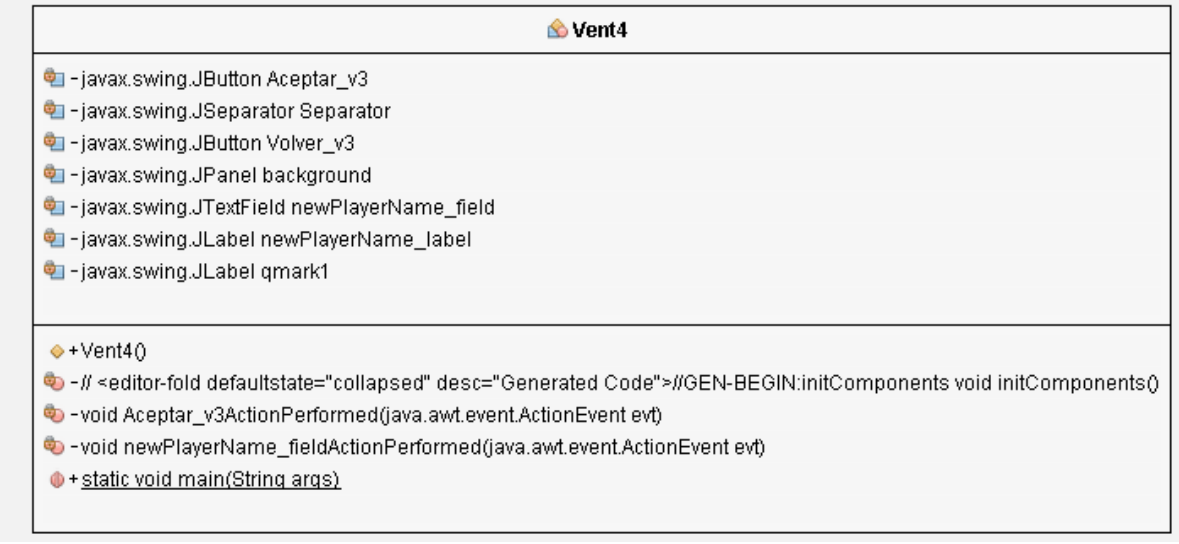
Diagrama UML de la clase Controlador, relacionada con DobbleGame.

A continuación, los diagramas de las ventanas de la interfaz, relacionadas entre ellas y con la clase Controlador mediante Vent2.




Vent2
~ static Controlador controlador
- javax.swing.JButton Crear_v3
- javax.swing.JSeparator Separator1
- javax.swing.JSeparator Separator2
- javax.swing.JSeparator Separator3
- javax.swing.JSeparator Separator4
- javax.swing.JButton Volver_v1
- javax.swing.JPanel background
+ static javax.swing.JTextField maxC_field
- javax.swing.JLabel maxC_label
+ static javax.swing.JTextField mode_field
- javax.swing.JLabel mode_label
+ static javax.swing.JTextField nPlayers_field
- javax.swing.JLabel nPlayers_label
+ static javax.swing.JTextField numE_field
- javax.swing.JLabel numE_label
- javax.swing.JLabel qmark1
- javax.swing.JLabel qmark2
- javax.swing.JLabel qmark3
- javax.swing.JLabel qmark4
+ Vent2()
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
- void Volver_v1ActionPerformed(java.awt.event.ActionEvent evt)
- void nPlayers_fieldActionPerformed(java.awt.event.ActionEvent evt)
- void maxC_fieldActionPerformed(java.awt.event.ActionEvent evt)
- void Crear_v3ActionPerformed(java.awt.event.ActionEvent evt)
- void mode_fieldActionPerformed(java.awt.event.ActionEvent evt)
- void numE_fieldActionPerformed(java.awt.event.ActionEvent evt)
+ static void main(String args)





En el laboratorio #3 la comunicación con el usuario se llevaba a cabo mediante la consola de comandos del sistema operativo, requiriendo una presentación clara, secuencial de los pasos a seguir para ejecutar la solución. Un aspecto que aún se mantiene de dicha implementación es la necesidad de comunicar las interacciones con el modelo mediante inputs y outputs de texto al usuario. Como este laboratorio no hace uso de la consola para interactuar con el programa, el feedback necesario durante una ejecución pasa a ser entregado en forma gráfica. Botones y campos de texto permitirán entregar al programa los datos necesarios para su ejecución, mientras que

paneles y “pop ups” de alerta comunicarán al usuario posibles errores, alertas e información relevante para el buen uso del programa.

Resultados

Del trabajo realizado, se pudo implementar una interfaz gráfica a modo de vista que interactúa con los modelos definidos durante el desarrollo de los laboratorios de programación orientada a objetos, avanzando en su ejecución mediante detección de eventos. A continuación, se encuentra la autoevaluación del laboratorio:

Puntaje a asignar (denotado en paréntesis junto a cada apartado):
0: No realizado.
0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
0.75: Implementación con problemas menores (funciona 75% de las veces)
1: Implementación completa sin problemas (funciona 100% de las veces)

1. Clases y estructuras que forman el programa (1)
2. Constructor de juegos (1)
3. Register (1)
4. Play (0)
5. toString() (0)
6. equals(Object o) (0)
7. vs CPU Mode (0)
8. demo Mode (0)

Como análisis de los resultados obtenidos, las clases diseñadas poseen los atributos para soportar ejecución de juegos dentro de la aplicación. Funciones “visualizar” emulan la utilidad de las funciones toString(), retornando información procesada por el modelo en forma de string para su fácil comunicación en pantalla.

Conclusión

Los objetivos de este laboratorio correspondían a poder integrar el paradigma orientado a objetos con el orientado a eventos mediante una interfaz gráfica que maneje el modelo diseñado en el laboratorio anterior. Dentro de los principales aspectos a tener en cuenta durante su diseño, se encuentra la dependencia unilateral entre Vista y Modelo, logrando que la Vista haga uso de los métodos definidos dentro de los Modelos, pero que a su vez este no tenga a la vista en su ámbito. Considerando lo anterior, se consideran logrados los objetivos para los requisitos funcionales trabajados en este laboratorio.

Referencias

1. Dore, M. (2021, 30 diciembre). *The Dobble Algorithm - Micky Dore*. Medium. https://mickydore.medium.com/the-dobble-algorithm-b9c9018afc52
2. A. (2020, 3 junio). *The Dobble Algorithm*. 101 Computing. https://www.101computing.net/the-dobble-algorithm/
3. Oracle. (s. f.). *Understanding Class Members (The Java™ Tutorials > Learning the Java Language > Classes and Objects)*. Oracle Java Documentation. Recuperado 10 de julio de 2022, de https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html