Proyecto: “Flappy Quetzal”
Versión 1.0

Descripción y Modelo de Análisis

Elaborado por:

- Cabello Díaz Sofía Elizabeth
- Lara Aguilar Christian Abraham
- Núñez Hernández Diego Ignacio
- Sánchez Hernández Carmen Alejandra

Fecha: 05/02/2021

1. Índice

1. *Índice* 2
2. *Introducción* 3
3. *Patrón de diseño* 3
4. *Diagrama de Clases* 4
6. *Diagramas de Secuencia* 7
5. *Casos de Uso-Análisis* 8
7. *Diagramas de Estado* 8
8. *Descripción de Arquitectura* 9
9. *Conclusiones* 9

Objetivo

El objetivo de este proyecto final es implementar todo lo aprendido en clase en la creación de un juego basado en FlappyBird.

Introducción

Es este trabajo se aplicaron los conocimientos de Java y de la Programación Orientada a Objetos para programar un juego de tipo endless runner basado en "Flappy Bird", que consiste en hacer volar a un quetzal para que esquive obstáculos.

Es un juego para un solo jugador incluye una interfaz gráfica donde el usuario puede mover al personaje principal (quetzal) hacia arriba y hacia abajo mientras el personaje vuela de manera constante.

La interfaz contiene obstáculos en forma de árbol. Cuando estos son tocados por el quetzal, muere. Cuando el jugador muere, el quetzal hace un sonido triste indicando el fin de la partida.

La principal funcionalidad que tiene el juego es un sistema de puntos que va aumentando conforme se van pasando los obstáculos y guarda los 3 mejores puntajes en un archivo.

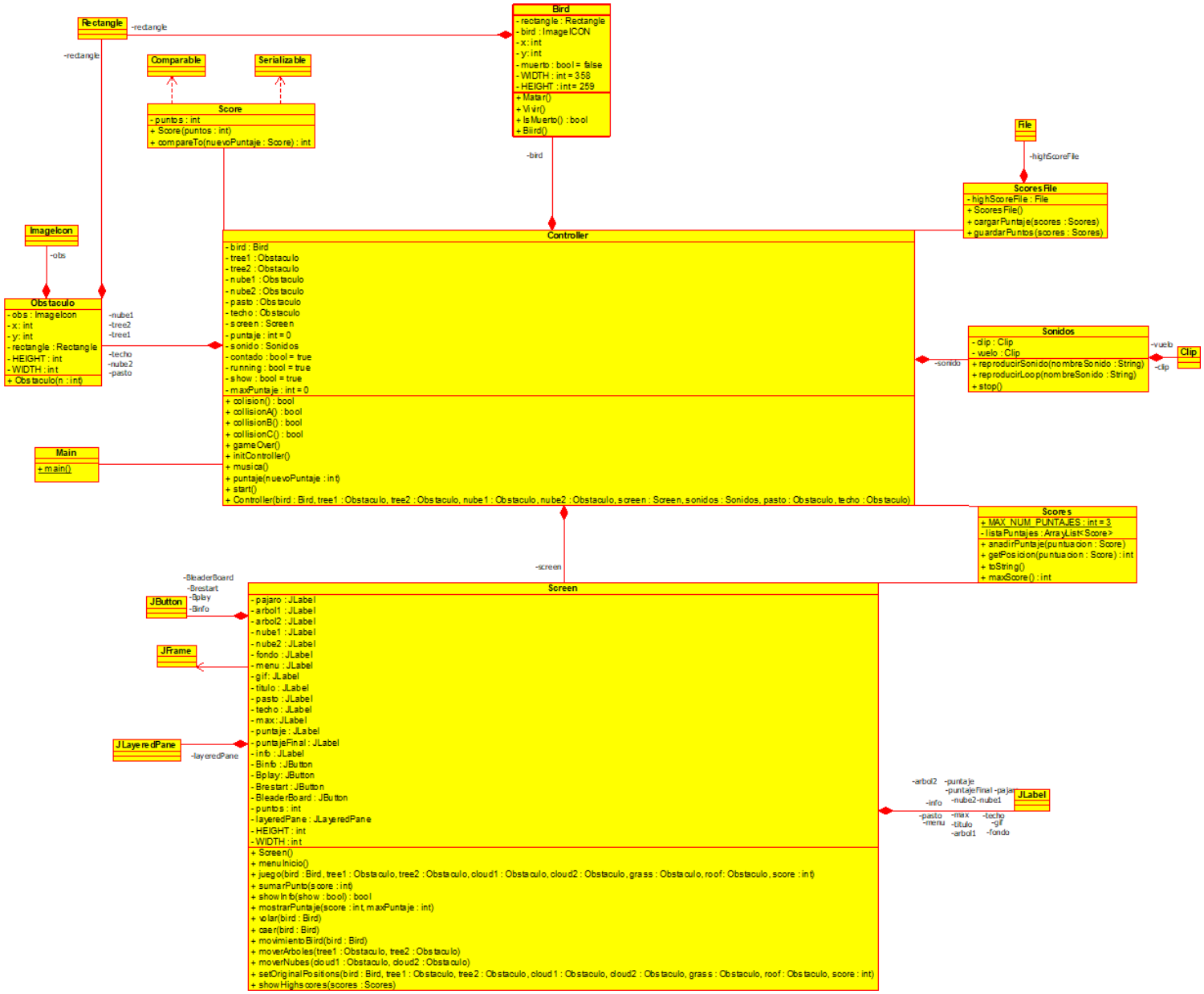
Patrón de Diseño

Los patrones de diseño han sido un activo valioso para desarrollar software por mucho tiempo ya que con ellos se escriben soluciones en un nivel medio de estructuras de software. Para crear nuestro juego desde cero se requirió hacer un diseño con el cual se pudiera trabajar la interfaz gráfica, con el propósito de facilitar esto se recurrió a uno de estos patrones.

El patrón de diseño usado para implementar el juego es el de Modelo-Vista-Controlador (MVC), con el que se le asignaron a todos los objetos tres roles diferentes. De manera que la parte del Modelo encapsula la información de los personajes, sonidos y del puntaje y define cómo se van manipular los datos con los que se está trabajando. La Vista se encarga de todos los elementos gráficos del juego, las ventanas, los paneles, imágenes y botones necesarios para llevar a la pantalla el juego, esto es precisamente, lo que ve el usuario cuando se ejecuta el programa, y que se actualiza según lo que él realice, ya sea iniciar una partida o ver las puntuaciones. Las acciones del usuario en la Vista son comunicadas al Modelo a través del Controlador, de tal manera que cuando un objeto del Modelo cambia, éste notifica al objeto del Controlador para que actualice los objetos de Vista necesarios y viceversa.

Como se puede ver en el diagrama de clases, la clase Controller tiene una relación de composición con todas las demás, puesto que en ella se relaciona todos los objetos en el programa.

Diagrama de clases



Modelo

En las clases que representan al modelo se define la lógica con la cual se van a poder manipular los datos de los objetos del juego, estas clases son Bird, Obstáculos, Scores, ScoresFiles, Score y Sonidos.

En la clase Bird se tienen todos los métodos y atributos relacionados con la imagen que representará al quetzal en pantalla, el ancho y alto de esta; sus coordenadas tanto en ex como en y; su estado, sea vivo o muerto; y un rectángulo, el cuál nos indicará sus bordes en todo momento. Esto último resulta útil para saber si ha habido colisiones o no respecto a otros elementos. En su constructor se asigna la imagen del quetzal y se define un rectángulo inicial, los demás métodos corresponden a los getters y setters necesarios para modificar estos atributos. De manera similar, todo esto se encuentra en la clase Obstáculos, la diferencia radica que mientras que en Bird siempre se usa la misma imagen, aquí hay cuatro opciones diferentes: árbol, nube, pasto o techo. Esto porque como indica el nombre, decidimos agrupar todos los obstáculos en la misma clase porque comparten los mismos atributos y métodos que el pájaro excepto por el booleano de estado, él cual ellos no necesitan.

Para manejar todo lo relacionado con los puntajes se usan tres clases diferentes. En la clase Score se tiene el puntaje del usuario al finalizar la partida, el cuál será comparado después con los demás. En ScoresFile se obtiene (o se crea en caso de no existir) el archivo highscorde.dat en el que se encuentran los tres puntajes más altos, tiene solamente dos métodos, en cargarPuntaje se obtiene la lista de puntajes previa con InputStream, mientras que con guardarPuntos se guarda, usando ahora flujos de salida. La lista de puntajes será modificada por la clase Scores, en la cual se tiene la lista previamente obtenida con el objetivo de hacerle cambios si son necesarios. Particularmente, el método añadir puntaje es el que se encarga de hacer o no una adición a la lista al usar la clase Score para comparar al nuevo puntaje, considerando que si ya hay tres valores previos y uno es menor que el actual, se tiene que eliminar el último. De igual forma se tienen métodos para imprimir su contenido y obtener el máximo puntaje en ella.

La clase Sonidos reproduce lo necesario para ambientar el juego. Tiene solamente tres métodos, reproducirSonido se encarga precisamente reproducir clips durante el juego, pueden ser la música de fondo, el haber obtenido un punto o al morir, por su parte reproducirLoop se encarga de que la tanto la música de fondo como el sonido del quetzal al volar no se detengan al acabar la pista. El método stop sirve para detener la música una vez que se termina la ejecución.

Controlador

En la clase Controller es donde se tiene el manejo de todo el juego. Tiene contacto con la Vista y con el Modelo, notificando a cada uno los cambios al programa, ya sean en pantalla o internos. En initController se cuenta con un mouseListener para saber si el usuario está dando clic en esta, o para saber si se ha presionado el botón de inicio.

Es con el método start con el que se lleva a cabo el juego. Se puede mencionar que en este método se instancia un hilo, una de las razones para haber hecho esto fue que los componentes se movían demasiado rápido para poder jugar, por lo que se tuvo que usar sleep para evitar esta situación. Sin embargo, no se quería detener la ejecución de todo el programa, puesto que lo único que en realidad se quería retrasar era la parte gráfica, detener todo ocasionaba errores en otros métodos dentro del controlador y la pantalla, por lo cual resultó conveniente el uso de un diferente hilo, el cuál se encarga de repetir el ciclo en el que ocurre el juego. En el run se le indica a Screen lo que tiene que mostrar en pantalla, es decir, que coloque los componentes y mueva a los árboles y nubes, así como

al pájaro si es que detecta que el usuario da clic en la pantalla. También se encarga de checar por colisiones, obtiene las posiciones de los elementos en otro método llamado collision, que a su vez llama a otros, de manera que devuelve verdadero si hay una intersección entre los rectángulos de algún obstáculo, ya sea árbol, nube, pasto o techo y el pájaro. Si esto no ocurre se sumarán puntos al usuario cada vez que pase un árbol y se repite el ciclo, de lo contrario, se cambia el estado del pájaro a muerto, se detiene la música y se llama al método gameOver.

En gameOver se guarda el puntaje y se muestra. Hace visibles a tres botones, en la esquina superior derecha aparecerá un botón de información en el cuál se muestran los integrantes del equipo; en el botón leaderboard se muestra una ventana con los tres puntajes máximos hasta el momento; con el botón restart se inicia una nueva partida, no sin antes iniciar el puntaje en cero y colocar a los componentes en sus posiciones iniciales.

Con respecto a la música se tiene un método que instancia un hilo para que siempre se esté reproduciendo cuando la ventana de juego esté abierta.

Vista

La principal clase que compone a la Vista es Screen. Es aquí donde vamos a encontrar todo lo que se muestra en la pantalla, cuando se instancia se crea la ventana en la que se van a colocar todos los componentes y se muestra el menú al inicio del juego, con su logo y el botón para empezar a jugar. Se utilizó la clase LayeredPane para poder colocar los elementos unos encima de otros, por ejemplo, que el fondo estuviera hasta atrás y que el pájaro estuviera enfrente de todo lo demás.

En su método juego se crean los JLabel para los elementos del juego, el pájaro, los obstáculos y el puntaje. Para hacer esto se obtiene la imagen de cada uno, se escala a la media adecuada y se coloca en una posición inicial dentro del frame. Más adelante, los métodos caer y volar permitirán que el pájaro se desplace, caer define que el pájaro siempre este moviéndose hacia abajo, mientras que volar permite que salte hacia arriba según lo indique el usuario. Como caer, los métodos moverArboles y moverNubes generan el movimiento de estos elementos, cambiando su posición periódicamente para dar la ilusión de desplazamiento en el juego. Para que los obstáculos no estén siempre a la misma altura se genera un número aleatorio que indica qué tan alto o bajo se colocará un árbol cada vez que otro haya salido de la pantalla, cada nube va asociada a un árbol y en base a él obtiene su altura, esto para que siempre sea posible para el pájaro superar los obstáculos.

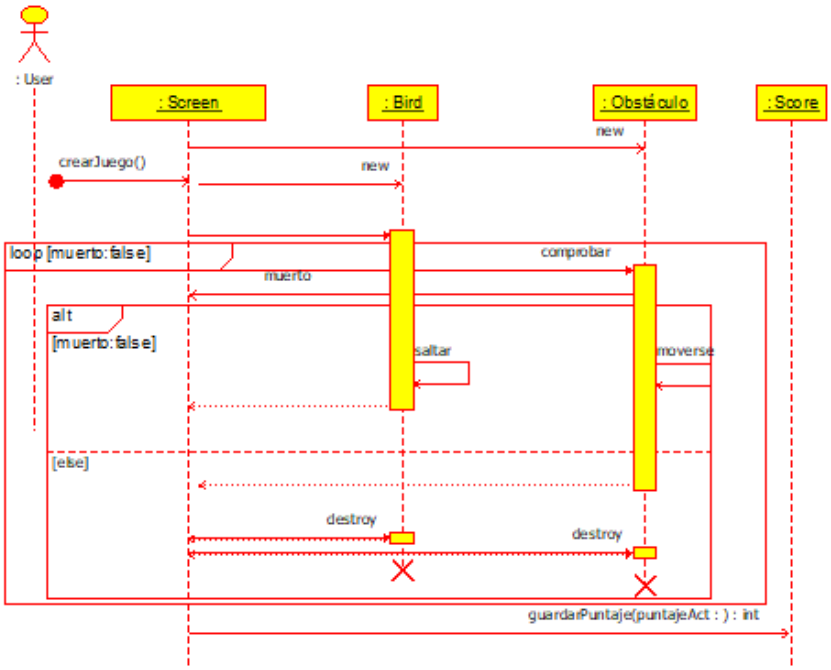
Otros métodos relevantes como mostrarPuntaje y showHighscores sólo se emplean una vez que el usuario ha perdido, ya que uno muestra los botones al final junto con el puntaje obtenido, y otro abre la ventana con los puntajes más altos. Para esto último se auxilia de la clase tableroPuntaje, que es la que instancia un frame nuevo y que imprime sus contenidos.

El método setOriginalPosition es el que regresa a todos los componentes a su posición inicial al terminar una partida.

Para que el pájaro esté siempre en movimiento se usa otro hilo que alterna entre las diferentes imágenes que se tienen del quetzal, como si fuera un gif, así se alcanza el efecto de que está volando durante el juego.

Algo importante a notar es que no se está haciendo un uso estricto del patrón de diseño, en la Vista se hacen algunos cambios directos al modelo, en específico a la posición de los personajes del juego, esto es necesario para tener un código más limpio y claro de entender, además de que estas actualizaciones permiten que el controlador pueda checar por colisiones durante toda la ejecución.

Diagrama de secuencia



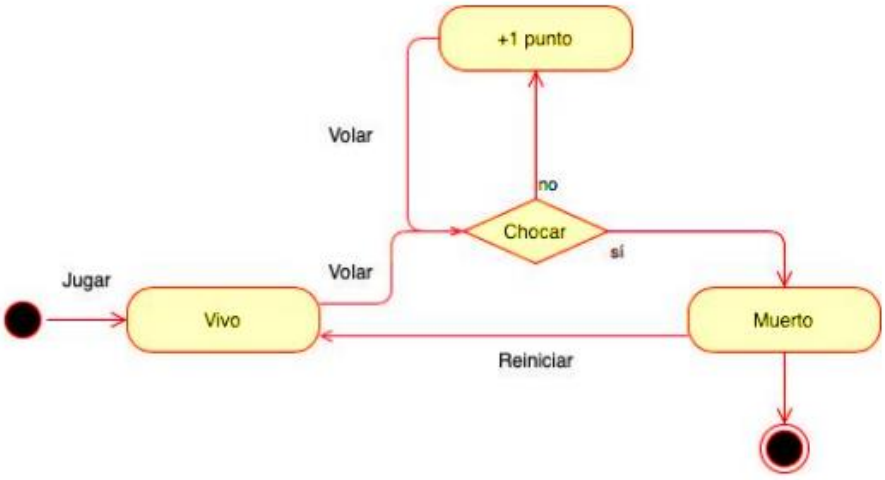
Este diagrama ilustra el momento en el que el usuario crea una partida. Primero se instancia un objeto de la clase Screen, así como de Bird y Obstáculos. Después se inicia un ciclo en el que se manda a llamar un método que sirve para comprobar las coordenadas del Pájaro y de los Obstáculos para saber si el personaje ha chocado o no. En caso de no chocar, el pájaro sigue saltando y los obstáculos se siguen moviendo hasta que las coordenadas de los dos objetos coinciden y se acaba el juego guardando el puntaje con la clase Score.

Casos de Uso-Análisis

Las funciones a las que puede acceder el usuario serán el acceso a los mejores puntajes y a crear una nueva partida cada vez que lo decida.

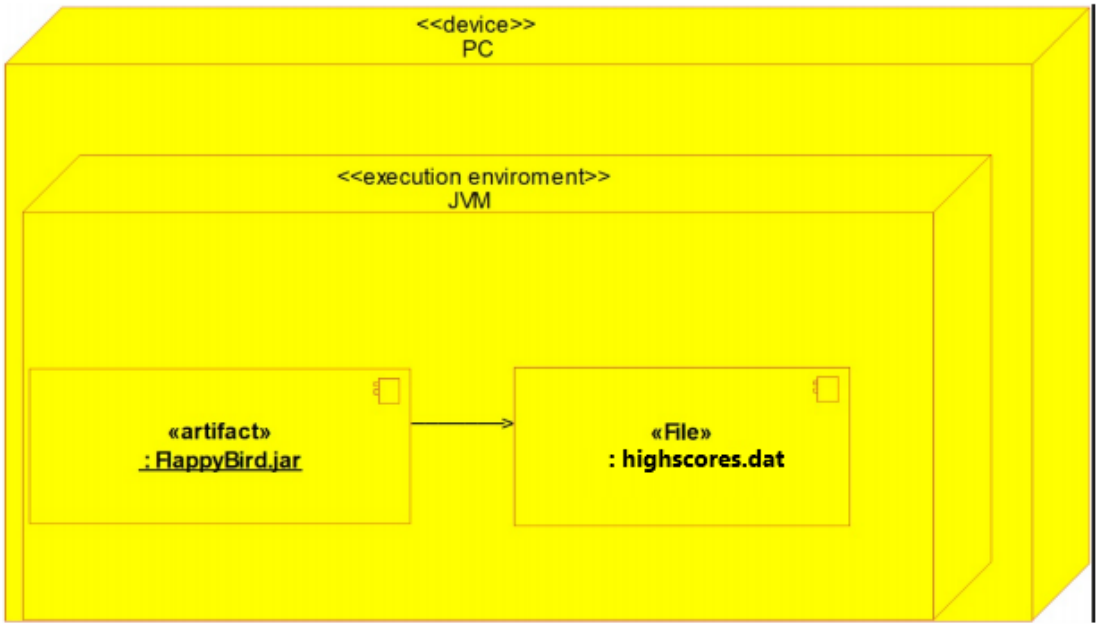
Diagrama de Estado

En este diagrama se ilustran los estados del objeto Pájaro durante la ejecución del programa. Al inicio del juego, el pájaro está vivo por lo que podrá estar volando. Cuando el objeto toca un obstáculo pasa a un estado de muerto hasta que el usuario decida reiniciar el juego, mientras que no choque, su puntuación aumentará en un punto y seguirá volando.



Descripción de Arquitectura

El juego será ejecutado en una computadora que tenga instalada la máquina virtual de Java. Dentro de esta máquina virtual se ejecutará el archivo .jar del programa y a su vez hará uso de un archivo para leer los puntajes del juego.



Conclusiones

El desarrollo de este videojuego nos permitió aplicar los conocimientos adquiridos a lo largo del curso de Programación Orientada a Objetos, puntualizando principalmente en el manejo de herencia, de paquetes, de archivos, así como el uso de los recursos, por ejemplo, de la clase Java Swing que nos permitieron y facilitaron la creación de una interfaz gráfica para poder interactuar con el usuario.

Consideramos que gracias a la elaboración de los diagramas UML (diagrama de clases, de estado, de secuencia y de usos) como parte del primer paso para el desarrollo del videojuego nos fue de mucha utilidad, ya que al estar trabajando en equipo nos permitió asignar adecuadamente las tareas que cada uno de nosotros debería realizar de acuerdo con lo establecido en los diagramas mencionados anteriormente, logrando poder unir cada una de las partes en las que se dividió el proyecto de manera exitosa.

Siendo este nuestro proyecto final de la asignatura nos percatamos de la importancia que esta tiene no solo en un aspecto académico, sino también abarcando un campo más amplio en nuestra vida profesional a futuro