UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

INGENIERÍA EN COMPUTACIÓN

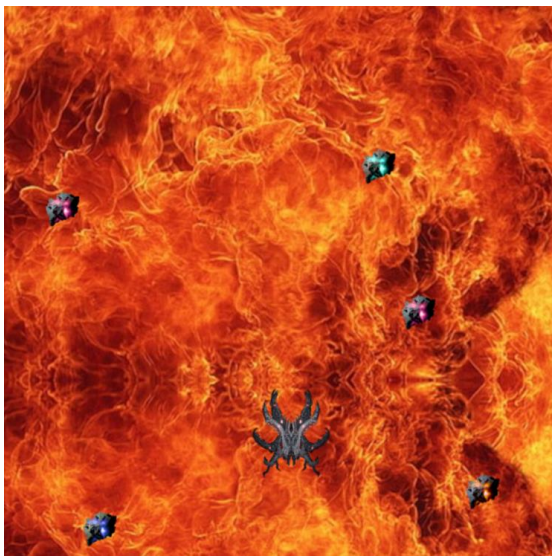
PROGRAMACIÓN ORIENTADA A OBJETOS

MIGUEL ANGEL SANCHEZ HERNANDEZ

VIDEOJUEGO MEJORE LAS FUNCIONES COGNITIVAS

ETERNAL SIN

LEONARDO OLVERA MARTÍNEZ



Índice

Objetivos

- Crear un videojuego a partir del modelo Canvas y JavaFX
- Implementar elementos de los videojuegos tales como personajes, puntajes usando las propiedades de java como lenguaje de programación orientado a objetos
- Usar el razonamiento lógico matemático para la ilusión del juego infinito

Introducción

La programación es el método por el cual el ser humano se “comunica” con la computadora, son una serie de instrucciones las cuales son escritas por un programador y la computadora las interpreta y ejecuta, pero no es tan sencillo como pedirle que haga algo “por favor”; la forma en que nos comunicamos con las maquinas es mediante los lenguajes de programación, algunos ejemplos de ellos son C++, Python, HTML, y Java, este último es el que utilizaremos para cumplir los objetivos planteados del proyecto.

Java es un lenguaje de programación orientado a objetos y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

La Programación Orientada a Objetos (POO) es una forma de programación que parte del concepto de "objetos" como base, los cuales contienen información en forma de campos y métodos.

Para lograr que un programa haga lo que queremos en POO, debemos darle una serie de instrucciones las cuales deben ir en código, a esto le llamamos métodos y estos se alojan en “plantillas” conocidas como clases y en ellas puede haber uno o más métodos, pero no se puede realizar un método si no tenemos datos y estos datos son conocidos como campos de clase o estados de clase. Las clases son capaces de interactuar con otras clases, pero para ello debemos usar objetos. Los objetos son capaces de interactuar y modificar los valores contenidos en sus campos o atributos y mediante constructores podemos crear una infinidad de objetos que contengan los mismos métodos, pero diferentes valores en sus atributos, siendo esta una de las versatilidades de la programación orientada a objetos. Muchas de las clases se pueden guardar en bibliotecas permitiéndole al usuario usar preestablecidas, las propias o de otros usuarios. Algunas características de la programación orientada a objetos son herencia, abstracción, polimorfismo y encapsulamiento.

El proyecto que se presentará a continuación tiene la finalidad de crear un videojuego utilizando los beneficios y las herramientas de Java como lenguaje orientado a objetos. Se ha planteado un videojuego infinito de tipo simulación el cual consista en una nave espacial escapando de un planeta a punto de estallar esquivando asteroides que se han generado por la naturaleza en destrucción del planeta.

Planteamiento del problema

Se ha demostrado que los videojuegos tienen un impacto positivo en las personas que los consumen, pues uno de sus mayores y mejores beneficios es que mejoran las funciones cognitivas de las personas. La demanda de videojuegos en la actualidad ha hecho que todo tipo de público quiera disfrutarlos, pero como se mencionó anteriormente, los beneficios que trae su consumo ha provocado que no sean solo medios de entretenimiento si no

también medios de aprendizaje y enseñanza en el campo pedagógico, esto es conocido como "gamificación". Lamentablemente los videojuegos diseñados para personas con capacidades diferentes son muy pocos y su demanda no es grande, además de ser un grupo históricamente relegado, el catálogo de videojuegos para ellos suele ser poco, y es importante cubrirlo.

Para darle solución se ha creado un videojuego el cual involucre la atención de quien lo juega. El planteamiento es sencillo para el jugador, una nave espacial que está esquivando asteroides que se va encontrando en su recorrido y a medida que el juego va avanzando la dificultad aumenta. La intención es que la persona que lo juegue mejore sus habilidades cognitivas, su tiempo de respuesta ante un suceso no previsto y además que sirva como medio recreativo para las personas con capacidades especiales.

Solución de la problemática

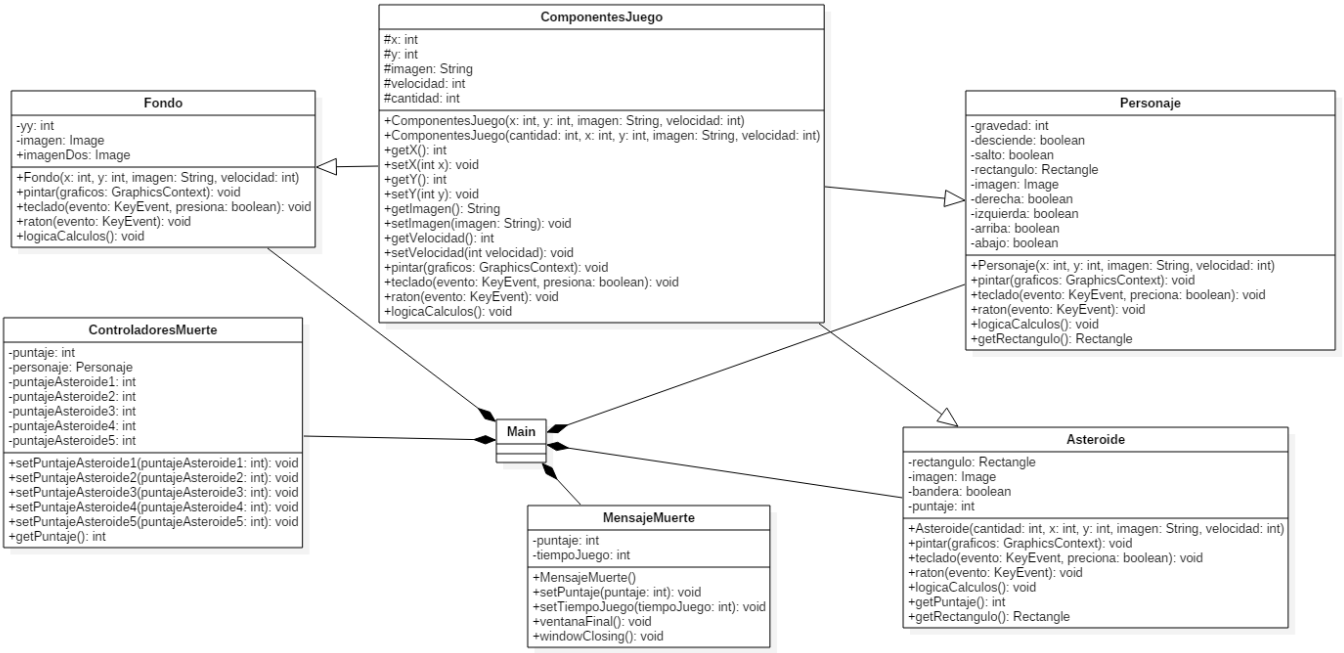
La poca cantidad de títulos de videojuegos para las personas con capacidades especiales genera un problema, demanda, y la solución para dicha problemática es desarrollar un videojuego que resulte recreativo e interactivo para las personas que necesiten de ello. La mayoría de las personas cuando se encuentran frente a un evento rutinario suelen aburrirse y buscar otra alternativa, en el caso de las prácticas de videojuegos en personas para capacidades distintas, no suele ser muy diferente, la única diferencia radica en que los títulos son poco variados para ellos, así que para cubrir esa necesidad es necesario crear más títulos para evitar una rutina. El videojuego debe tener controles sencillos, pocas mecánicas, debe ser visualmente atractivo y tener un factor que muestre el avance como tiempo o un puntaje y un grado de dificultad para que las personas, ante ello, lo consideren un reto.

Desarrollo de la solución

UML

Creamos un modelo UML que nos ayudará como plano para la realización del proyecto.

En el se encuentran todas las clases que se tienen planeadas para el proyecto: los componentes del juego que será “el papá” de distintas clases usando herencia y esta clase definirá las funcionalidades de las ubicaciones y los eventos de teclado, ratón y una lógica de cálculos; después tenemos las clases para el personaje principal, las naves y el fondo, las cuales heredaran las funcionalidades de la clase de los componentes del juego; posteriormente las clases que se encarguen de mostrar un mensaje de muerte y finalmente la clase Main la cual llamará a todas las anteriormente mencionadas



ComponentesJuego

Lo primero que se realizó en el proyecto fue la creación de la clase Main, posteriormente la creación del fondo, pero para ello hacemos la lógica que regirá el videojuego. Creamos una clase llamada ComponentesJuego la cual nos ayudará con la posición en todos los elementos mostrados en pantalla. ComponentesJuego tendrá 5 atributos con encapsulamiento protected (ya que la finalidad de la clase es que las demás clases que hereden de ella); los atributos son “x” y “y” (los cuales nos ayudaran a manipular los objetos en lienzo; “imagen” la cual nos ayudará a cargar del paquete “fes.aragon.recursos” las imágenes queremos plasmar; y finalmente “cantidad” y “velocidad” que son dos tipos de variables enteras que nos ayudaran a darle una lógica matemática a nuestros elementos. Después agregamos dos constructores para futuros objetos: uno para el personaje y el fondo y el otro para los asteroides. Agregamos los métodos set y get para los campos de clase “x”, “y”, “imagen” y “velocidad” y finalmente importamos las bibliotecas de JavaFx “javafx.scene.canvas.GraphicsContext” y “javafx.scene.input.KeyEvent” y agregamos 4 métodos para saber si el usuario está presionando una tecla, usando el mouse, para pintar los gráficos y para realizar la lógica de cálculos (lo que se realice en estos métodos será definido en posteriores clases).

Canvas, JavaFX y los gráficos

Una vez que tenemos la clase ComponentesJuego podemos comenzar a programar el fondo, pero para ello necesitamos crear un espacio donde la aplicación pueda pintar las cosas. Nos dirigimos a la clase Main para crear un componente Canvas. Importamos las bibliotecas de JavaFx, nos permitirán manipular el Canvas

javafx.scene.Group	Con esta biblioteca crearemos un campo de clase llamado “root” que usaremos para agregar una hoja a la clase escena
javafx.scene.Scene	Usando esta biblioteca crearemos un objeto de tipo escenario con un campo de clase llamado “escena” usando los valores de “root” y las dimensiones de 600x600
javafx.scene.canvas.Canvas	Crearemos un atributo llamado “hoja” y su respectivo objeto con las dimensiones de 600x600 y lo agregaremos a la función getChildren() de “root”
javafx.scene.canvas.GraphicsContext	Creamos un atributo llamado “graficos” y lo igualamos al objeto con los valores del método getGraphicsContext2D() del objeto “hoja”

Los atributos serán globales para poder usarlos en cualquier parte de la clase, por lo tanto, deben estar afuera de cualquier método. En cambio, los constructores los meteremos dentro de una clase pública llamada “componentesIniciar()” la cual ejecutará todos los elementos los gráficos del videojuego.

javafx.stage.Stage	Esta biblioteca nos permitirá crear la ventana para la aplicación. Creamos un método público llamado “start” que tenga un objeto “ventana” de la clase Stage, para esto debemos heredar de “Aplicattion” de la biblioteca de JavaFX “javafx.application.Application”. Dentro del método se encontrará la clase “componentesIniciar()” para que llame a este método y contenga los gráficos. Después agregaremos los métodos de “ventana”: setScene() y los cambiamos por el objeto “escena”; setTitle() y agregamos un título para el escenario y finalmente show() para mostrar el escenario.

Ya tenemos el escenario configurado, solamente nos falta mostrar algo en él. Para ello crearemos una clase llamada “Fondo” y esta heredará de ComponentesJuego. Necesitamos un escenario que se mueva, pero crear un escenario infinito podría presentar problemas con algunos equipos ya que serían datos infinitos los cuales almacenar, por lo tanto, usaremos una lógica matemática y de programación para hacer una ilusión que el escenario es infinito. Creamos dos atributos de clase llamados: “imagen” e “imagenDos”. Creamos un constructor Fondo con los atributos “x”, ”y”, ” imagen”, ” velocidad” (heredados por la clase ComponentesJuego). Del atributo “imagen” que recibiremos en el constructor lo igualamos al atributo de la clase “imagen” y para no crear otra clase mediante el método replace() cambiamos el “1” que será la primera imagen y lo cambiamos por “2” que será la misma imagen, pero volteada y esto lo igualamos con el atributo imagenDos. Creamos los métodos que heredamos para no romper la lógica del programa: pintar(GraphicsContext graficos) y agregamos los métodos drawImage() de ambas imágenes, creamos un campo de clase “yy”, lo igualamos a -600 y se lo asignamos a “imagenDos” para que se posicione antes de “imagen”; teclado(); ratón() y finalmente logicaCalculos() donde “y” y “yy” le

sumaremos el valor de velocidad que obtenemos del constructor, después usamos una lógica matemática para que una vez el valor de “y” y “yy” sea igual a 600 lo iguale a -600 para que se encuentre en un bucle infinito que cree la ilusión de un mapa infinito usando únicamente dos imágenes. Finalmente agregamos el constructor al método componentesIniciar() para crear un objeto en la clase Main.

Para poder verlo en pantalla crearemos un método privado en la clase “Main” llamado: “pintar” en el cual agregaremos los métodos de los objetos de cada elemento que queramos ver en pantalla.

Personaje

Lo siguiente que tenemos que hacer es crear al personaje principal: la nave que escapará de los asteroides. Creamos una clase en el paquete “fes.aragon.modelo” con el nombre de “Personaje”. Heredamos de la clase ComponentesJuego para usar las coordenadas, los eventos de teclado y los métodos “pintar” y “logicaCalculos”. Creamos los atributos de clase privados: “gravedad” y la igualamos a 1 (la cual en la logicaCalculos” la usaremos); “imagen” de tipo Image y “rectangulo” de tipo Rectangle, para hacer esto debemos importas las bibliotecas “javafx.scene.shape.Rectangle” y “javafx.scene.image.Image”; creamos 5 atributos de clase tipo booleano llamados “salto”, “derecha”, “izquierda”, “arriba” y “abajo”, a todos les asignamos el valor false (nos servirán para los movimientos del personaje. Después creamos un constructor con los atributos que heredamos de la clase ComponentesJuego. Igualamos la imagen que recibiremos en el constructor con el campo de clase “imagen” para poderla usar y después creamos un objeto rectangulo usando a “x” y “y” que recibimos en el constructor y le asignamos el ancho y alto con los métodos “getWidth()” y “getHeight()” del objeto “imagen”. Posteriormente agregaremos los métodos heredados: “pintar()” donde haciendo uso de la biblioteca (previamente importada) “javafx.scene.canvas.GraphicsContext” haciendo uso del método “drawImage()” le daremos los valores de “imagen”, “x” y “y” para que pinte al personaje con la imagen y las coordenadas que le daremos en el constructor; el método “teclado()” y la biblioteca “javafx.scene.input.KeyEvent” podremos saber si el usuario está presionando alguna tecla,

así que si es así revisaremos con un “if” que tecla está presionando y llamaremos al atributo de clase booleano correspondiente a esta acción y lo igualaremos a verdadero (en el caso que el usuario presione la tecla arriba llamamos al atributo “arriba” y le asigna el valor verdadero) pero para evitar un bucle infinito y la nave suba infinitamente creamos un else el cual, si no se está presionando alguna tecla asigna el valor de los atributos a false, esto para que no siga una dirección de manera infinita; agregamos el método ratón(); después agregamos el método “logicaCalculos()” en donde a “y” le vamos a estar agregando constantemente el valor de “gravedad”, el cual es 1, para que cree la ilusión que la nave tiene algún tipo de retroceso, posteriormente con un if, para cada atributo booleano, comprobaremos si su valor es verdadero y de ser así moveremos la nave usando lógica de programación. En caso de que sea verdadero el atributo “derecha” a x le sumará el valor de velocidad; en caso de “izquierda” restará a x el valor de velocidad; en caso de arriba y salto restará en y (salto será 5 lo que reste); en caso de abajo sumará a y el valor de velocidad. Después asignaremos los valores de “x” y “y” del objeto rectangulo con los valores de “x” y “y” del personaje usando los métodos “setX” y “setY”. Finalmente, para este método haremos que la nave no se salga del escenario. Mediante el condicional if comprobaremos si los valores de los laterales son menores a 0, si es así le sumaremos la velocidad para cancelar este movimiento (en el caso de los laterales derecha y abajo, como las coordenadas son del primer pixel en la esquina superior izquierda lo que haremos es sumar el valor de “x” o “y” según sea el caso más el ancho o alto de la imagen usando los métodos del objeto imagen “getHeight” o “getWidth” y comprobando que no sean mayores que 600). Finalmente, para esta clase, agregamos un método “getRectangulo” que devuelve un valor de tipo Rectangulo.

Nos vamos a la clase Main y creamos un objeto para poder crear un personaje en la aplicación. Agregamos el objeto al método pintar.

Asteroide

Lo siguiente es crear los asteroides. No se puede crear un juego de esquivar obstáculos sin obstáculos. Creamos una clase en el paquete “fes.aragon.modelo” que se llamará “Asteroide” y heredamos de ComponentesJuego para hacer uso de sus métodos. Creamos dos atributos privados: “rectangulo” de tipo Rectangle e “imagen” de tipo Image. Después crearemos un constructor el método que heredamos de ComponentesJuego con los valores de: “cantidad”; “x”; “y”; “velocidad” (int) e imagen (String), tenemos un problema y es que una vez que creemos un objeto de tipo “asteroide” y le demos un valor en “x” y “y” no se lo podremos cambiar así que dentro del mismo constructor haremos que los valores de “x” y “y” sean aleatorios pero nos encontramos frente a otro problema ya que puede que desde el inicio todos estén formando una muralla imposibilitando el paso, así que usaremos el valor de “cantidad”, este nos indicará cual es el turno del asteroide para mostrarse en pantalla. Tomamos el valor de cantidad y mediante “if” y “else if” sabremos si será el asteroide número 1, 2, 3, 4 o 5 y si es 1 le asignaremos al valor de “y” 0; si es 2 le asignaremos -150; si es 3 le asignaremos -300; si es 4 le asignaremos -450 y si es 5 le asignaremos -600, pero nuevamente nos encontramos frente a un problema, los asteroides bajarán con una velocidad constante por lo tanto la distancia entre asteroide y asteroide siempre será la misma, así que para solucionar esto creamos dos variables enteras: “min” y “max” y en el “if” y los “else if” en lugar de igualar “y” con esos valores, le asignaremos el valor (dependiendo de que numero sea el asteroide) a la variable “min” y posteriormente le asignamos el valor de “min” – 50 a “max” y creamos una función que nos de un numero aleatorio entre esos números y se lo asignamos a “y”. Después, usando la misma lógica le asignamos el valor de un numero aleatorio entre 0 y 550 para que no ocurra el mismo problema que se encuentren todos bajando en una misma posición en “x”. Creamos un rectangulo y le asignamos los valores de “x”, “y”, el largo y el ancho de la imagen. Agregamos los métodos de ComponentesJuego: “pintar” y haciendo uso de la librería de JavaFX “javafx.scene.canvas.GraphicsContext” le damos el valor al método con esta librería llamándolo “graficos” y usando le método “drawImage” pintamos el asteroide; agregamos los métodos “teclado”; “ratón” y “logicaCalculos”. En este ultimo lo que haremos será configurar a los asteroides para que bajen, pero no podemos crear infinitos objetos de tipo asteroide así que usando una lógica matemática haremos que una vez que “y” sea igual a 600 le reste 700 a “y” posicionándolo nuevamente al inicio del mapa, nuevamente nos dé un valor aleatorio en “x”, cambie el valor de “puntaje” sumándole 1 al valor anterior y “bandera” a true (un campo de clase que igualaremos a false). Cuando “bandera” sea true le sumará 1 a la velocidad con la que baja el asteroide y volverá a cambiar el valor de “bandera” a false, pero hay un problema: ahora que la velocidad ha cambiado puede que bajando el asteroide pase de 599 a 601, siendo esto así jamás volverá al inicio y con el tiempo ya no habrá asteroides, así que cambiamos ese “if(y == 600)” por un “if(y >= 600” para que en estos casos cualquier asteroide que salga de pantalla vuelva arriba. Finalmente agregamos los métodos “getPuntaje” y “getRectangulo” los cuales usaremos después para ver si la nave principal ha chocado con un asteroide y saber el puntaje una vez que el jugador haya perdido.

Nos trasladamos a la clase “Main” y creamos 5 objetos de tipo asteroide en donde cada nuevo objeto tendrá un turno distinto empezando en 1 y terminando en 5 para conservar la lógica que programamos en la clase “Asteroide”. Agregamos los métodos correspondientes de los 5 asteroides en el método “pintar” de la clase “Main”.

Movimiento

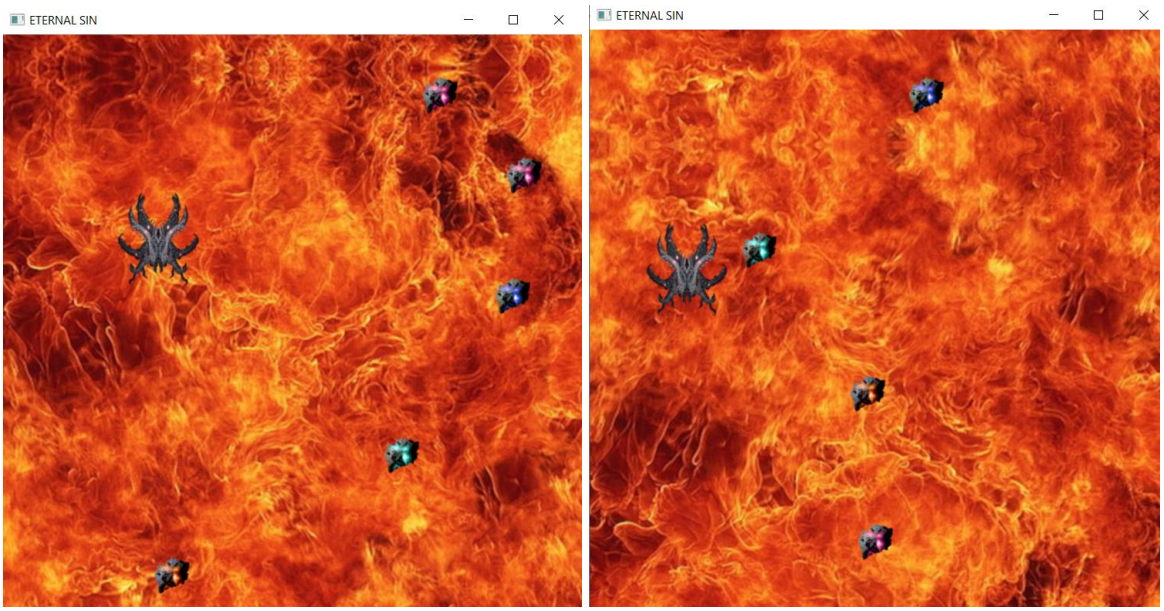
Hasta ahora la aplicación funciona a medias ya que, aunque muestre los elementos como la nave o el fondo, por más que presionemos alguna tecla nada pasa, así que lo siguiente que tenemos que encargarnos es de el movimiento. Ya tenemos la lógica del movimiento y como funciona cada objeto, solo falta implementarlo en la clase “Main”. Creamos un método llamado “calculosLogica” en el cual implementamos los métodos correspondientes de la lógica de los movimientos de los objetos que serán los elementos que tengan una animación en el videojuego tales como el fondo, la nave y los asteroides. Ahora debemos saber si la nave choca con algún asteroide por lo que lo que haríamos seria comparar si no ha chocado y si choca deje de hacer los cálculos. Creamos un “if else” y dentro del “if” ponemos el objeto “personaje” junto con los métodos “getRectangulo” seguido de “getBoundsInLocal”, esto para obtener el rectangulo del personaje principal, después ponemos el método “intersects” y entre el paréntesis pondremos el objeto del primer asteroide y, siguiendo la lógica que usamos para obtener el rectángulo del personaje, obtenemos el rectangulo del objeto del primer asteroide, de esta manera sabremos si la nave ha chocado con el primer asteroide. Seguimos este paso para cada uno de los asteroides dentro del “if” para que revise si no ha chocado con el primero o el segundo, etc. En “else” ponemos los métodos para la lógica de cálculos que anteriormente agregamos de esta manera si choca con un asteroide no haga nada y mientras no sea así, puede continuar con los cálculos, siendo así, cuando choque la aplicación se quedará pausada. Lamentablemente la aplicación sigue sin funcionar como teníamos previsto ya que el método “pintar” no actualiza lo que está haciendo el método “calculosLogica”.

FPS

La aplicación no está refrescando las imágenes así que dentro de la clase “Main” crearemos el método público “ciclo” en donde crearemos dos temporizadores, uno el cual iniciará al momento de ejecutar la aplicación y otro que será llamada por cada frame que sea ejecutado en la aplicación. Los restamos y dividimos entre 1000000000 y lo que obtenemos será el tiempo de ejecución de la aplicación, este valor lo igualamos a un campo de clase que posteriormente llamaremos a una ventana emergente al morir, después agregamos los métodos “calculosLogica” y “pintar” para refrescar la imagen. Finalizamos el método iniciando el objeto de tipo “AnimationTimer”.

Agregamos el método “ciclo” en el método “start” para que así no se quede sin refrescar.

Para hacer más dinámico el juego meteremos música. Agregamos la canción a la biblioteca “fes.aragon.recursos” en el método “componentesIniciar” creamos un objeto de tipo “MusicaCiclica” (Esta biblioteca la importaremos en una nueva carpeta llamada “fes.aragon.extras”). Creamos un objeto “Thread” con el valor del objeto de “MusicaCiclica” y después lo iniciamos. Creamos un método para detenerlo cuando cerremos la aplicación y también lo detenemos cuando la nave impacte con un asteroide.



Muerte

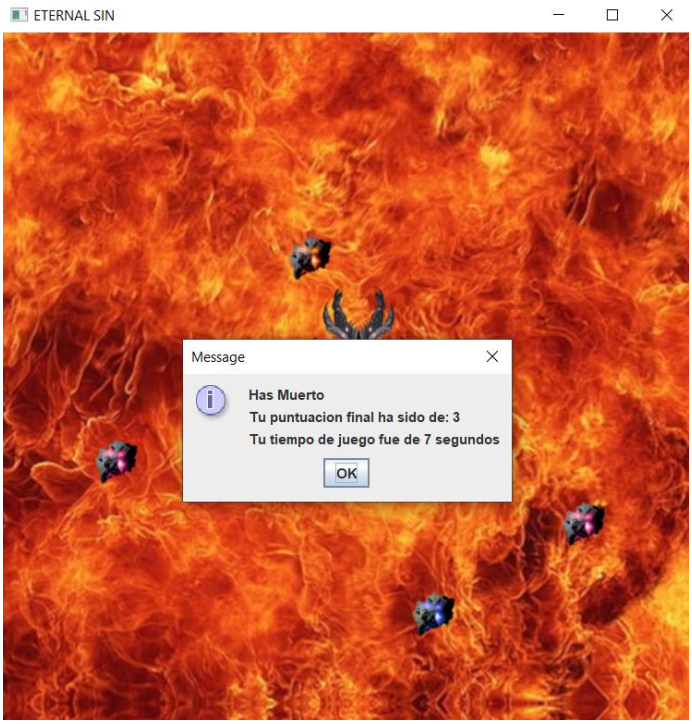
Ahora que se ha detenido el juego, como programadores, sabemos que hemos muerto, la nave ha chocado con un asteroide, pero el usuario no sabe esto, todo lo contrario, puede llegar a pensar que se a trabado el juego o ha hecho algo mal, así que lo que debemos hacer es encontrar una forma para enseñarle al usuario que lo que ha pasado es que ha muerto.

Crearemos una ventana emergente que diga que ha muerto, mostraremos su puntaje y el tiempo que ha sobrevivido y la crearemos de tal forma que una vez las animaciones se detengan muestre la ventana con los datos y al cerrar la ventana se cierre el juego.

Creamos dos clases en el paquete “fes.aragon.modelo”: “ControladoresMuerte” y “MensajeMuerte”. En la clase “ControladoresMuerte” lo que haremos será obtener el puntaje total de cada asteroide (recordando que son 5). Creamos 6 campos de variable enteras privadas, una para el puntaje y las otras 5 para cada puntaje de los asteroides. Creamos los métodos “set” para cada campo, los mandamos llamar en la clase “Main” en él método “calculosLogica” y les cambiamos el valor por sus respectivos asteroides. Cuando cada campo tenga el valor de sus asteroides, creamos un método “get” para el puntaje y ahí mismo igualamos el puntaje con la suma de cada atributo de los puntajes de los asteroides.

Creamos una clase para la ventana llamada “MensajeMuerte”. Creamos dos atributos, uno para el puntaje y otro para le tiempo. Creamos los métodos “set” de ambos atributos y desde el método “Main” en el método “calculosLogica” le asignamos el valor del tiempo en la variable que creamos para el método “ciclo” y para el puntaje la variable que creamos anteriormente. Finalmente creamos dos métodos: el primero es para crear un “Jframe” y en él, usando la función “JOptionPane.showMessageDialog” mostraremos el tiempo y el puntaje usando los campos de clase de estos anteriores; el ultimo método se encargará de cerrar la aplicación el cual usará un método “exit”

Agregamos estos dos últimos métodos al final de las acciones que se tomarán cuando la nave se impacte con un asteroide y hemos terminado.



Conclusiones

JavaFX nos permite como desarrolladores de Java crear aplicaciones de una manera eficiente y sencilla.

Uno de los mas grandes retos de un programador es que funcione su lógica de programación, pues por mas que conozca bibliotecas y métodos, si la lógica de su programa es incorrecta el programa puede no funcionar. En el proyecto creamos la ilusión de un videojuego infinito, el cual conforme va avanzado se hace mas difícil y todo esto fue gracias a una buena lógica de programación, además que nos ahorra trozos de código.

Java como lenguaje de POO es el rey. Miles de programadores de especializan en Java por su versatilidad en este tipo de programación. Una de las facilidades que nos proporciona y fueron usadas en el proyecto fue la herencia, encapsulamiento y los objetos que nos ayudaron a manipular fácilmente los asteroides y la manera en que se usó la animación en los elementos del videojuego.

El videojuego fue creado y todos los objetivos planteados al inicio del proyecto fueron completados.

La versatilidad de Java como lenguaje orientado a objetos y JavaFX como herramienta de gráficos y aplicaciones son indispensables para los programadores que desean realizar alguna aplicación, pues Java esta la mayoría de los dispositivos en el mundo. De la misma manera un gran número de las aplicaciones son creadas con JavaFX, siendo una de las herramientas más útiles al momento de diseñar una aplicación.

Bibliografía

Deitel, P. & Deitel, H. (2008). *Como programar en Java*. Pearson Educación de México.

MuyInteresante (2018). *5 beneficios de jugar videojuegos según la ciencia.* [Artículo]. Recuperado el 20 de Mayo de 2022 de: https://www.muyinteresante.com.mx/ciencia-tecnologia/beneficios-jugar-videojuegos/

Oracle. (-). *What is JavaFX?.* [Artículo]. Recuperado el 20 de mayo de 2022 de: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm