|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Ing. Guadalupe Lizeth Parrales Romay |
| *Asignatura:* | Programación Orientada a Objetos |
| *Grupo:* | 04 |
| *No de Práctica(s):* | 9 |
| *Integrante(s):* | Dávila Ortega Jesús Eduardo - No. Cuenta: 317199860  Díaz Hernández Marcos Bryan - No. Cuenta: 317027253  Pareja Ávila Emiliano - No. Cuenta: 317081345  Vázquez Zavala Oliver Alexis - No. Cuenta: 317202263 |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* | 12,14, 30, 37 |
| *Semestre:* | 2021-1 |
| *Fecha de entrega:* | 12 de diciembre del 2020 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_

**Objetivo de la práctica.**

Utilizar UML como herramienta para diseñar soluciones de software para un lenguaje de programación orientado a objetos.

**Introducción.**

UML, por sus siglas en [inglés](https://www.ecured.cu/Ingl%C3%A9s), Unified Modeling Language es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software, en él se muestran los estados y las interacciones entre objetos dentro de dicho sistema, en el caso de la programación orientada a objetos los diagramas UML se utilizan para representar los siguientes componentes del sistema:

-Objetos individuales (elementos básicos)

-Clases (combina elementos con las mismas propiedades)

-Relaciones entre objetos (jerarquía y comportamiento/comunicación entre objetos)

-Actividades (combinación compleja de acciones/módulos de comportamiento)

-Interacciones entre objetos e interfaces

Los diagramas UML se dividen en dos principales tipos, son estáticos, también llamados estructurales, a los que se encargan de definir qué elementos (entidades, objetos, áreas, clases, departamentos, componentes etc.) deben de estar definidas dentro del sistema u organización a desarrollar el correspondiente modelado y en dinámicos a los que se encargan de dar la vista de la relaciones e integraciones de los de los tipos de clases u objetos en ejecución del sistema.

* Diagramas estáticos:

-Diagrama de Clases: muestra las clases (descripciones de objetos que comparten características comunes) que componen el sistema y cómo se relacionan entre sí.

-Diagrama de Objetos: Los diagramas de objetos modelan las instancias generadas a través de las clases y se utilizan para describir al sistema en un instante de tiempo (o acción) en particular. Permiten mostrar los objetos y las relaciones entre ellos en un momento dado, por lo tanto, representa la parte estática de la interacción entre objetos (una situación específica en un momento determinado).

-Diagrama de Casos de Uso: modela la funcionalidad del sistema agrupándola en descripciones de acciones ejecutadas por un sistema para obtener un resultado. Se utiliza para entender el uso del sistema, en él se muestran el conjunto de casos de uso y actores (un actor puede ser tanto un sistema como una persona) y sus relaciones, es decir muestra quién puede hacer qué y las relaciones que existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

-Diagrama de Despliegue (o implementación): Los diagramas de implementación permiten visualizar la arquitectura física del hardware, el software y los artefactos del sistema. Los diagramas de implementación pueden entenderse como lo contrario de los casos de uso, porque ilustran la forma física del sistema, en lugar de representar conceptualmente los usuarios y dispositivos que interactúan con el sistema.

* Diagramas dinámicos:

-Diagrama de Estados: Se utiliza para analizar los cambios de estado de los objetos. Muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.

-Diagrama de Actividades: Es un caso especial del diagrama de estados, simplifica el diagrama de estados modelando el comportamiento mediante flujos de actividades. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

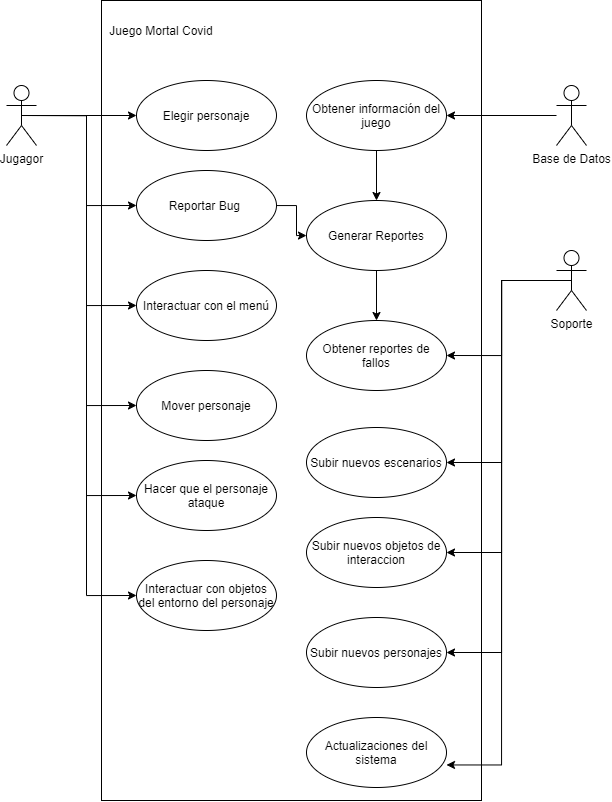
-Diagramas de Interacción: estos diagramas representan situaciones dinámicas y en particular, son adecuados para modelar el comportamiento en el que los elementos intercambian información, estos diagramas definen el papel de los objetos implicados, también nombran y priorizan los mensajes que se envían de un lado a otro entre los objetos. Los diagramas de interacción, a su vez, se dividen en otros dos tipos:

1.- Diagrama de Secuencia: enfatiza la interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.

2.- Diagrama de Colaboración: igualmente, muestra la interacción entre los objetos resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.

**Análisis de los ejercicios.**

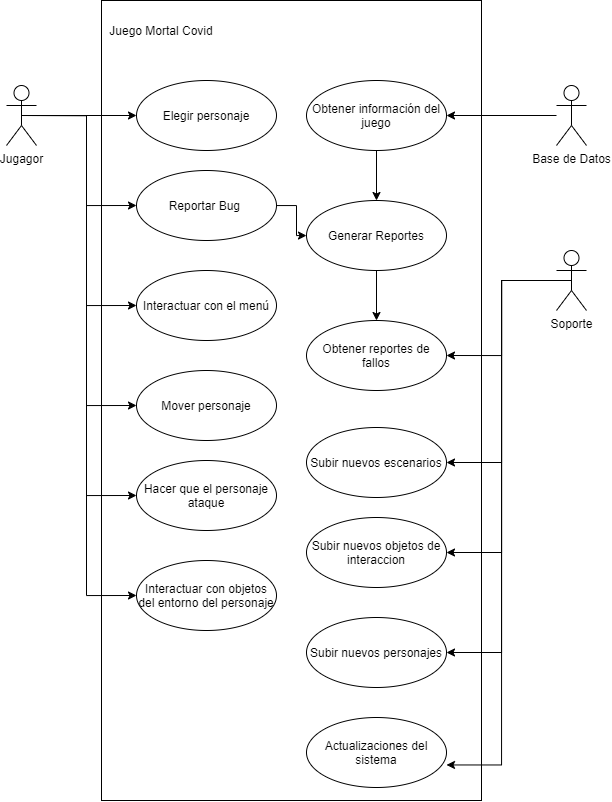
* **Casos de uso**

****

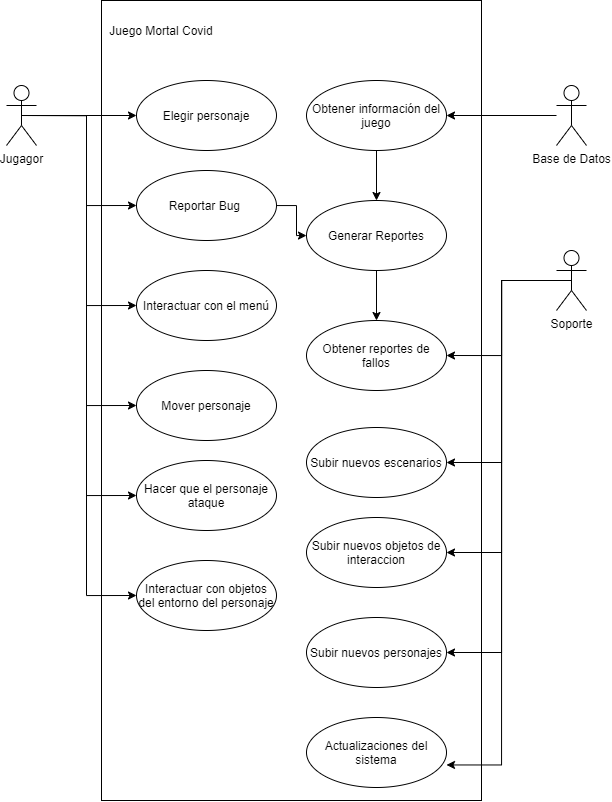
Este diagrama es el primero que se necesita hacer para poder modelar los requerimientos del usuario, y las funciones que tendrá nuestro sistema, este diagrama se compone de elementos importantes que provienen del mismo análisis del programa, estas son las limitantes que tiene el programa en cuanto a que se necesita llevar a cabo en este, y qué operaciones deben de llevar los actores.

Las entidades que componen el diagrama son:

* Casos de uso: son las funciones que el sistema debe de proveer de tal manera que esta produzca un resultado que se pueda visualizar por el actor, de igual forma funciona como una descripción del sistema.

****

* Actores: es una entidad externa que va a interactuar con el sistema

****

Como tal el sistema se representa por medio del rectángulo, donde se coloca la etiqueta correspondiente al sistema que representa, y donde se colocan los casos de uso dentro del sistema y los actores fuera de este.

Para sistema, que denominamos Juego Mortal Covid, se crearon diversos casos de uso para poder llevar a cabo el análisis completo del sistema, a sí mismo se establecieron tres actores que van a interactuar:

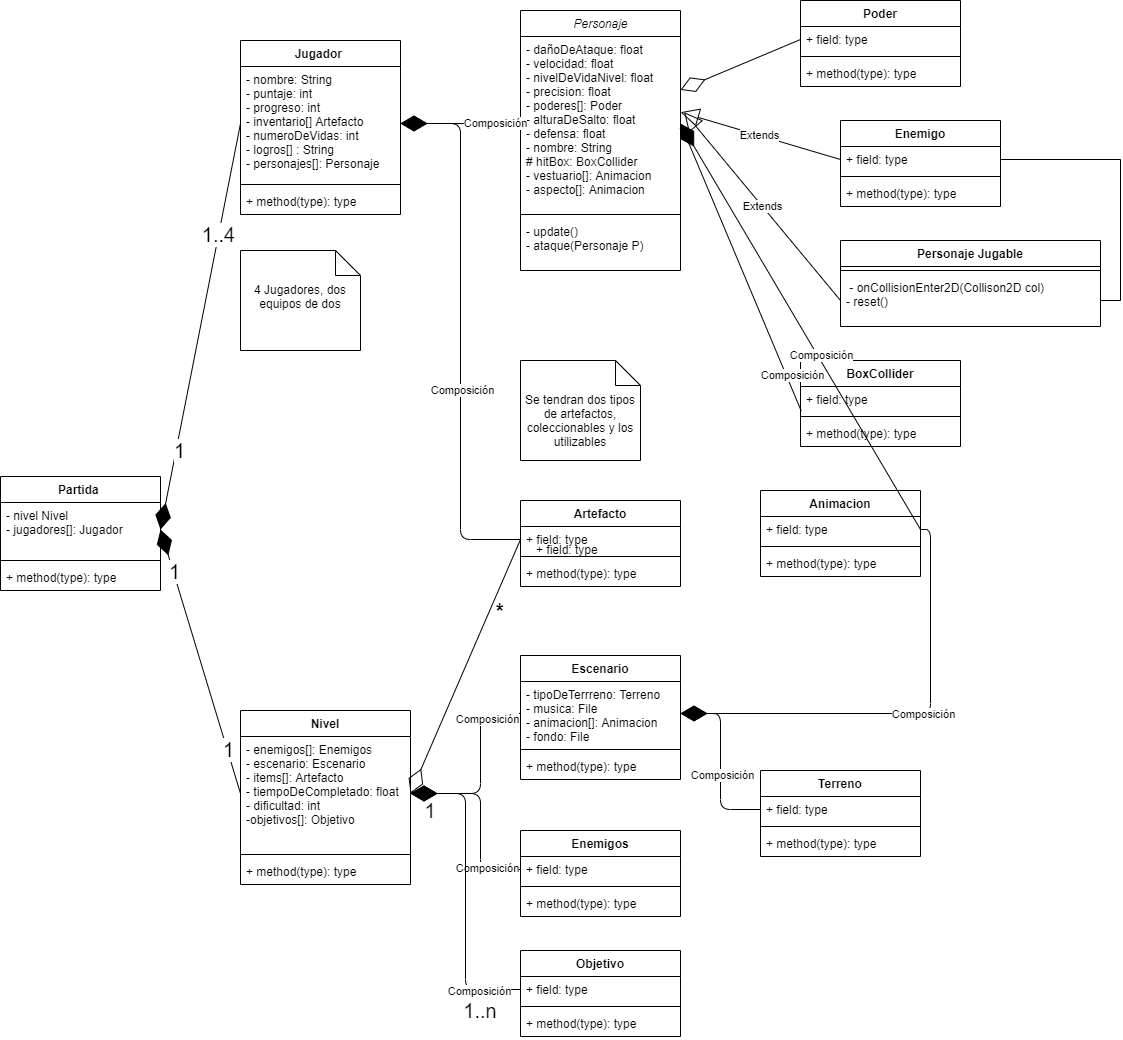
**Jugador:** el cual es quien interactúa más con el sistema de tal manera que este utiliza los siguientes casos de uso:

* Elegir personaje: por medio de este caos de uso se tiene el actor jugador, va a elegir un personaje cuando ya se ha iniciado una partida, de tal forma que va a tener una selección personajes que puede escoger.
* Reportar Bug: el caso realiza reportes cuando ocurre un error, un fallo, o un bug dentro del sistema, después de reportarse un fallo este se envía a generar reportes y cuando el tercer actor interactúa con el sistema se obtienen estos reportes que se generaron.
* Interactuar con el menú: en caso de que el actor quiera seleccionar una de las opciones que el permite el sistema se tiene que puede acceder a un menú para seleccionar guardar, cargar o salir del sistema.
* Mover personaje: cuando el actor está dentro de una partida debe poder mover al personaje, con el periférico que el actor haya elegido y la selección de controles que este prefiera.
* Hacer que el personaje ataque: dentro de una partida el personaje seleccionado tendrá que defenderse, para esto tendrá que eliminar a los enemigos por medio de ataques diversos, como a melee, o con artefactos.
* Interactuar con objetos del entorno: al estar jugando un nivel será necesario el poder recorrer el mapa, y en este habrá elementos que se podrán recoger y utilizar por el personaje.

**Base de datos**: que va a interactuar para obtener la información del juego, en el momento que un usuario reporte errores, en caso de que se desee guardar o continuar desde un determinado momento del juego.

**Soporte**: es el actor que se encargará del mantenimiento del sistema, para dar solución a los errores, a los bugs, y para actualizarlo.

* Obtener reportes de fallos: en caso de que el jugador, haya hecho reportes, el soporte deberá obtener estos reportes y resolverlos, por medio de las actualizaciones del sistema que le permitirán poder resolver cualquier tipo de problema con el sistema.
* Subir nuevos escenarios: en el caso de las actualizaciones y del soporte al jugador se debe de subir nuevo contenido que permita mejorar la experiencia del actor jugador, tal y como lo son los niveles que puede pasar el jugador.
* Subir nuevos objetos de interacción: en el caso de las actualizaciones y del soporte al jugador se debe de subir nuevo contenido que permita mejorar la experiencia del actor jugador, tal y como lo son nuevos artefactos que poder utilizar y con esto permitir que los niveles cambien.
* Subir nuevos personajes: en el caso de las actualizaciones y del soporte al jugador se debe de subir nuevo contenido que permita mejorar la experiencia del actor jugador, tal y como lo son nuevos personajes que poder seleccionar e ir avanzando en nivel con el personaje.
* Actualizaciones del sistema: las actualizaciones permiten que la integridad del sistema se conserve y haya mejoras a la par para la solución de problemas que se reporten además de que en estas se añade de igual forma nuevo contenido con las subidas de nuevos elementos.
* **Diagramas de clase**

****

Para este diagrama de clases, primero desarrollamos la clase Jugador, que es la que modela a un jugador de nuestro videojuego, y le dimos los atributos nombre de tipo String, puntaje y progreso que son de tipo entero, un arreglo del tipo de dato Artefactos, que recibe por nombre inventario, otro atributo llamado númeroDeVidas que es de tipo entero, un arreglo de la Strings que llevan por nombre logros y un arreglo de personajes del tipo de dato Personajes. Después desarrollamos la clase personajes, ya que estos son muy importantes para el juego, porque a partir de estos podemos jugar en el juego de batalla. La clase Personajes tiene una relación de composición con la clase Jugador, ya que para que un jugador pueda jugar, valga la redundancia, necesita un arreglo de personajes con los cuales jugar y poder combatir en el videojuego, cabe señalar que todos los atributos de nuestro personaje serán de carácter privado, ya que como desarrolladores de videojuegos, si alguien accede a los atributos de esta clase podría modificarlos y hacer que su progreso sea más avanzado o que su puntaje dentro del videojuego cambie y sea muy alto.

Después se desarrolló la clase Personaje, que es una de las principales para nuestro videojuego, dentro de esta cuenta con los atributos danoDeAtaque que es de tipo float, y es el daño que aplica el personaje al momento de atacar, velocidad que también es un float y que es la velocidad del personaje al momento de avanzar en el nivel, otro atributo es nivelDeVidaNivel que es de tipo float, y determina con cuánta vida cuenta el personaje dentro del nivel, tomando como referencia a las barras de vida que aparecen en los videojuegos de combate similares, otros atributos de tipo float son la altura de salto, la precisión de ataque y la defensa, y el nombre es otro atributo de la clase que es de tipo String, luego, desarrollamos los atributos que son de un tipo de clase, uno se llama poderes y es un arreglo de objetos de la clase Poder, otro atributo llamado HitBox, que es de la clase Boxcollider, y finalmente 2 atributos de la clase Animaciones llamados vestuario y aspectos, que serán un arreglo de animaciones. Y en sus métodos de la clase instanciamos solo 2 métodos aunque podrían ser muchos más, uno de estos se llama ataque y recibe como parámetro un objeto de la clase Personaje que será el personaje al cual atacará, y el otro método se llama update, que básicamente se ejecutará cuando el desarrollador del videojuego decida cargar nuevas actualizaciones al juego.

Entonces al tener varios atributos que componen a la clase Personaje, decidimos desarrollarlas, la primera es la clase Poder, y aunque no desarrollamos por falta de tiempo los atributos y métodos de esta clase, dentro de ella pudimos instanciar algunos atributos que incluyeran animaciones que hicieran lucir el poder del personaje. También creamos la clase BoxCollider, aunque también no la pudimos desarrollar, esta ayudará a que el personaje sepa cuando está chocando con alguna entidad del videojuego o lo están atacando. Y finalmente creamos la clase Animacion que dentro de ella tendrá métodos y atributos que nos permitirán modelar y generar un conjunto de animaciones que permitan ver cómo se mueve un personaje o cómo ataca este mismo.

Después hicimos un análisis sobre los personajes del videojuego, ya que dentro de él, habrá entidades que se enfrentan, y las dos entidades son los personajes que puede utilizar un jugador y los enemigos que este se enfrenta, entonces hicimos una relación de herencia de la clase Personaje, ya que tanto un personaje jugable y un enemigo son personajes, pero con diferentes objetivos. Dentro de la clase PersonajeJugable, desarrollamos 2 métodos, uno es un método que un compañero de nuestra clase que ha programado videojuegos nos referenció, que es de Unity y se llama onCollisionEnter2D que con la Hitbox del personaje permite al propio conocer su entorno y poder saber si está chocando con alguna entidad del escenario o con algún otro personaje. Y el otro método se llama reset y este permitirá resetear los atributos de un personaje jugable, en esta clase también se pudieron desarrollar más atributos y métodos de un personaje jugable, pero no pudimos generar más de estos por el tiempo, aquí se pudo desarrollar una sobrecarga del método atacar y que este reciba como parámetro un enemigo. Y después desarrollamos la clase Enemigo, que aunque sus métodos serán bastante similares al de un personaje jugable, también se puede hacer una sobrecarga del método atacar y que este reciba como parámetro un objeto de la clase PersonajeJugable.

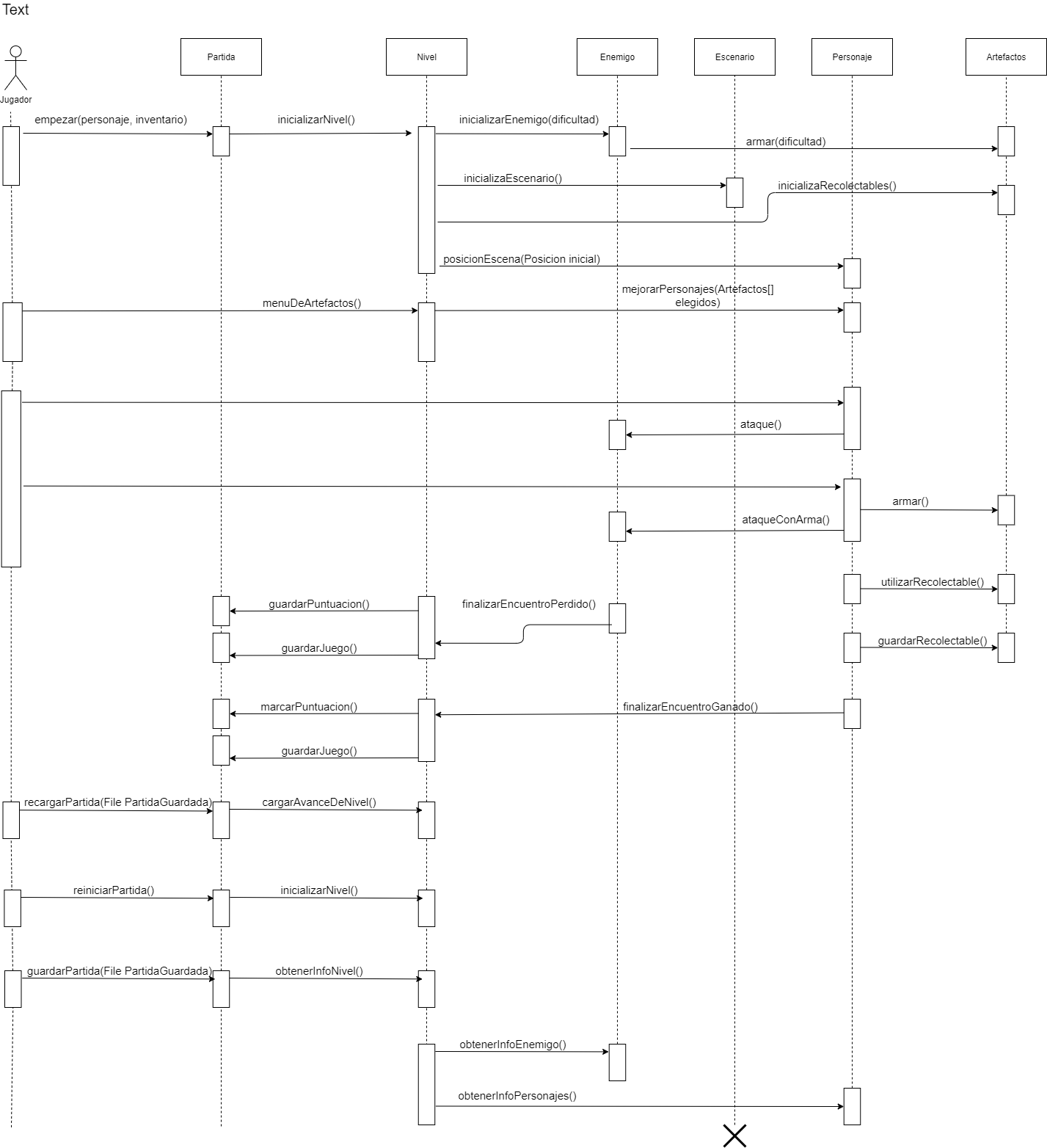
Una vez que concluimos en desarrollar todas las clases que se componen o se derivan de la clase Personaje, creamos dentro del diagrama de clases la clase Artefacto, esta es bastante importante, ya que hay diferentes artefactos que pueden ser utilizados por un jugador, para que éste incremente las características de su personaje como su nivel de vida, que incremente el daño de ataque que hace, etc. Y de esta clase Artefacto ser derivarán 2 clases que son la clase Coleccionable y Utilizable. La clase Coleccionable serán artefactos que no pueden ser utilizados para incrementar los atributos de un personaje, pero podrían sumar puntos de experiencia o podrían ser canjeados por un artefacto utilizable. Y la clase Utilizable nos servirá para mejorar las características de un personaje que utilice un jugador, por ejemplo se pueden utilizar espadas, armaduras o armas que ayuden al personaje a tener un mejor rendimiento y dentro del nivel de juego, se pueden crear artefactos de esta clase que ayuden a que el personaje utilizado por el jugador recupere vida.

Volvimos a hacer un análisis sobre las características básicas de un juego de batalla, y aunque modelamos los atributos y métodos básicos de una clase Jugador, aun nos faltaba el entorno donde se desenvuelve el juego, ya que sin un escenario no podría haber una buena experiencia de juego. Entonces creamos la clase Escenario y creamos sus atributos. Uno de estos recibe el nombre de tipoDeTerreno y es de la clase Terreno, esta clase nos permitirá modelar el terreno en el cual se desarrollara el juego dentro del escenario, por ejemplo, puede ser un escenario con un terreno lleno de nieve, de tierra o incluso de asfalto, depende de cómo lo queramos hacer. Otro atributo se llama música y este será un objeto de la clase File, y nos permitirá que durante el juego dentro del escenario podamos escuchar algún tema que se relacione con la situación en la cual se desenvuelve el juego. El tercer atributo que se creó se llama animación y es un arreglo de objetos de la clase Animación, esta se creó, ya que nuestros escenarios incluirán varias animaciones en el fondo que permitan una buena experiencia de juego, y finalmente creamos el atributo fondo que es un objeto de la clase File, ya que será una imagen que estará en el fondo de nuestro escenario, y tiene que ser de la clase File para poder cargarla desde una base de datos.

Se volvió a hacer un análisis sobre las características básicas de un juego de combate, y basándonos en otros videojuegos de este estilo, varios de ellos utilizan niveles para que el jugador pueda enfrentar mejores enemigos o que avance en los objetivos del juego en general. Por ende creamos en el diagrama la clase Nivel, que tendrá varios atributos, y se pueden observar varias relaciones de composición con clases que ya se crearon, por ejemplo un atributo se llama enemigos y será un arreglo de objetos de la clase Enemigo esto es necesario ya que por nivel se deben de tener varios enemigos, se tiene otro atributo llamado escenario que es un objeto de la clase Escenario y como lo indica por su nombre, este atributo permitirá mostrar el escenario en el cual se desarrollará el juego dentro del nivel. Se creó otro atributo llamado ítems que será un arreglo de objetos de la clase Artefacto, estos objetos ítems se podrán utilizar por el jugador para que su personaje recupere vida o que pueda incrementar alguno de sus atributos como su defensa o daño de ataque y estos ítems aparecerán a medida que se avanza en el juego o se derrota a un enemigo. Otro atributo creado recibe el nombre de tiempoDeCompletado y será un contador que tendrá el jugador para acabar el nivel, otro atributo llamado dificultad, que será la dificultad en la cual se desarrollará el nivel, que al ser de tipo entero, asumimos que esta puede ir de un rango de 1 a 5, donde 1 es la dificultad más fácil y la 5 la más difícil. Y creamos otro atributo llamado objetivos que es un arreglo de objetos de la clase Objetivo, y aunque creamos la clase Objetivo, no creamos ningún atributo o método dentro de esta, pero uno de estos métodos podría decidir la recompensa por cumplir uno de estos objetivos o que verifique si un objetivo se cumplió. Entonces, ya se tiene lo básico para modelar un juego, que son jugadores, personajes, escenarios y niveles, pero faltaba englobar todo esto en una sola partida de juego, ya que para poder jugar debemos de tener una partida que inicialice todo, desde el nivel, los personajes, enemigos, etc.

Por tanto creamos la clase Partida que tiene solo 2 atributos, uno se llama nivel y es un objeto de la clase Nivel, y otro atributo llamado jugadores que es un arreglo de objetos de la clase Jugador, y por partida podemos tener 4 jugadores repartidos en 2 equipos, o sea que por equipo puede haber máximo 2 jugadores. Y culminamos así nuestro diagrama de clases, faltaron desarrollar varias clases, pero cumple con las características más básicas de un videojuego de combate, también se podría crear otra clase que modele el juego en general y que uno de sus atributos sea partidas, y que sea un arreglo de objeto de la clase Partidas, ya que normalmente un videojuego de combate son un conjunto de partidas las cuales se tienen que superar con éxito para lograr algún objetivo.

* **Diagramas de interacción**

****

Para el diagrama de interacción se modela de forma gráfica los objetos de las clases que van a interactuar y los métodos que van a ser utilizados en cada una de estas interacciones entre objetos, además ayuda a identificar la jerarquía y orden de las clases, así mismo permite descubrir carencias en el modelado de las clases y en los casos de uso, ya que es posible crear nuevas clases y nuevos casos de uso.

En este caso se crea un diagrama de interacción para nuestro diagrama de casos de uso, además que se implementa y se basa en el diagrama de clases para saber qué objetos van a interactuar. Por medio de los mensajes es que se pueden enviar los mensajes a los objetos de cada clase y de esta manera llevar a cabo los métodos respectivos para cada objeto y para cada mensaje.

El análisis fundamental es saber qué objetos van a interactuar y qué mensajes va a recibir cada objeto, es básicamente lo que se desarrolla en el diagrama de clases, ya que por medio de este se modelan las interacciones entre las clases, y se sabe cual es la composición de una clase con otra.

Las entidades que caracterizan a este diagrama son los:

* Objetos de las clases: estos son los elementos que van a interactuar por medio de los mensajes entre ellos.
* el tiempo de interacción: esta es la medida que indica cuánto dura cada mensaje entre los objetos.
* El actor: en este caso se utiliza un actor debido a que los mensajes entre los objetos se comienzan a llevar a cabo a partir de la selección del actor.

En el diagrama se presenta el mensaje empezar(personaje, inventario), como la primera opción del actor, que para este caso estaría iniciando una partida desde cero y sería necesario indicar al objeto de Partida que se empiece una partida, posteriormente se tiene que enviar un mensaje al objeto de Nivel, que se inicialice un nivel. En base a esto los objetos de Enemigo, Escenario, y Artefactos recibirán un mensaje de Nivel para que estos se inicialicen, de acuerdo con esto el objeto de Enemigo de enviará armar(dificultad) al objeto de Artefacto para que este pueda armar a los enemigos en base a la dificultad que se haya elegido en el mensaje de inicializarEnemigo(dificultad).

De igual forma Nivel enviará posicionEscena(Posición inicial) al objeto de Personaje para poder colocar al personaje seleccionado dentro del nivel, una vez enviados todos los mensajes correspondientes entre los objetos de las clases, estará listo el juego para comenzar a jugarlo, y poder recorrerlo.

Durante la partida será posible el acceder a un menú, para acceder a este el sistema deberá de enviarle menúDeArtefactos() al objeto de Nivel, y Nivel a la vez deberá enviarle mejorarPersonajes(Artefactos[] elegidos), a Personaje, de tal forma que sea posible el mejorar a los personajes que el actor eligió, y mejorarlos en base a los artefactos que estén disponibles en el objeto de artefactos.

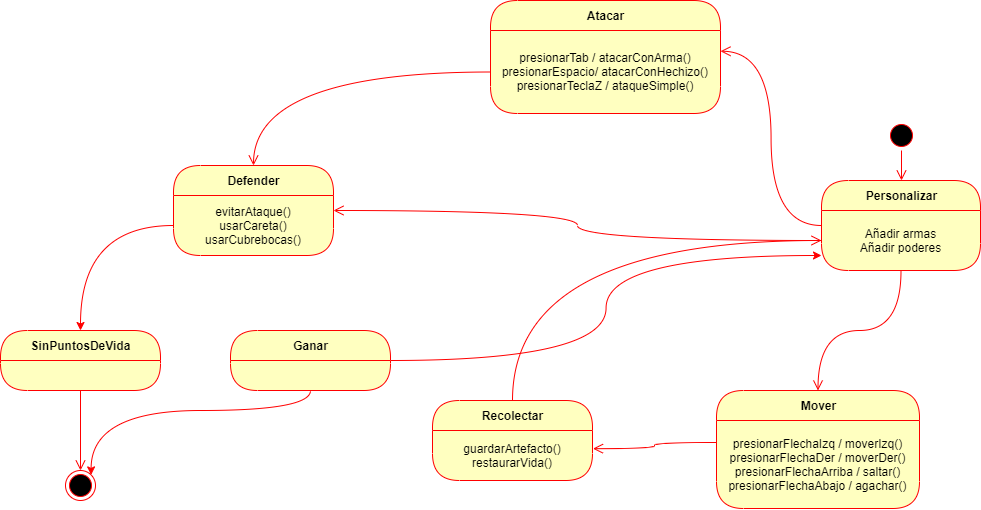
Para los ataques del personaje y de los enemigos se envía un mensaje al objeto de Personaje para que este ataque al objeto de Enemigo y de esta forma exista un golpe de nuestro personaje hacia un enemigo. De igual forma se puede atacar al enemigo, pero para hacerlo con un arma es necesario el armar al personaje, y para ello se envía armar() al objeto de Artefacto, lo que resulta en armar a nuestro personaje y posteriormente se envía el mensaje ataqueConArma() al objeto de Enemigo.

Para esta última interacción entre los objetos hay dos eventos consecuencia, que es ganar el combate o perderlo. Para el primer caso se envía un finarlizarEncuentroPerdido() desde Enemigo, hasta Nivel, para que se guarde la puntuación en Partida y se guarde el progreso de la partida. Para el segundo caso se envia finarlizarEncuentroGanado() a Nivel y se repite el proceso de guardar los datos en Partida. Un mensaje libre es utilizarRecolectables() porque este mensaje se puede enviar en cualquier momento de la partida, desde Personaje a Artefactos.

En caso de que el actor vaya a cargar una partida antigua, se envía el mensaje recargarPartida(File PartidaGuardada) a Partida y se envía cargarAvance() a Nivel, en caso de reiniciar una partida existente o actual, se envía reiniciarPartida() a Partida e inicializarNivel() a Nivel.

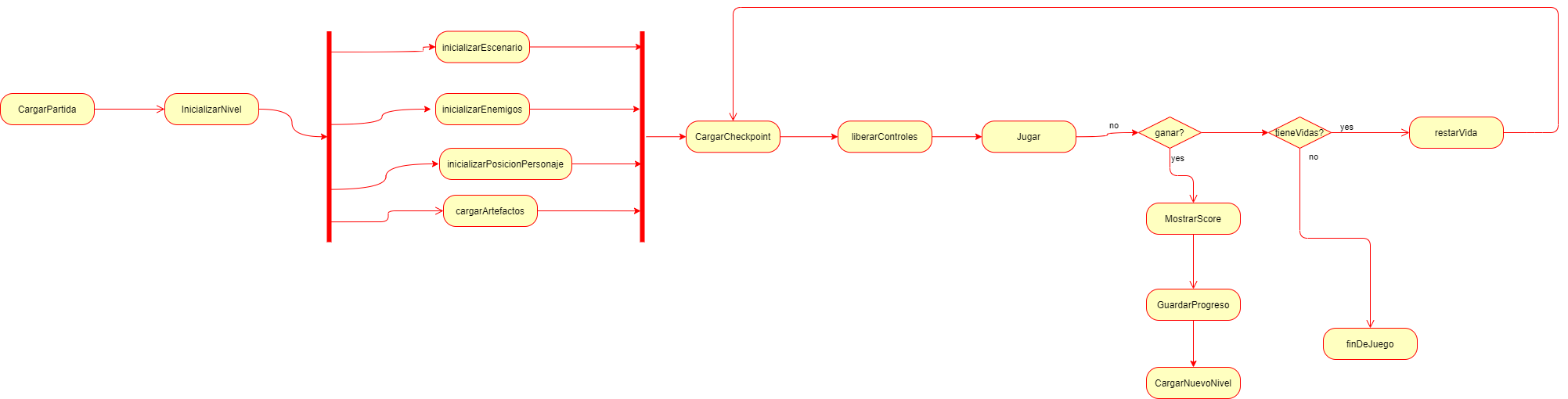
Para el caso de guardar la partida se envía GuardarPartida(File PartidaGuardada), a Partida y se envía obtenerInfo() a Nivel, Enemigo, Personajes, para que los datos sean guardados en el File, y así conservar el progreso del actor.

* **Máquinas de estado**

****

La máquina de estados define en qué estados puede estar el programa en un momento dado y cómo estos se interrelacionan, el estado inicial del programa es al momento de personalidad, ya que al principio de un nivel lo primero que se hace es elegir las habilidades y/o armas que deseas equipar, después del estado de personalización existen cuatro estados que están relacionados con la personalización, uno de estos es atacar, ya que nuestro ataque va a estar dado por las armas y/o poderes que hayamos seleccionado previamente, asimismo ocurre lo mismo con el estado de defender, ya que para defendernos algunas veces se necesitarán recurrir a los elementos seleccionados al momento de personalizar, otro estado ligado a la personalización es el de mover, ya que la rapidez con la que se desplaza el personaje va a estar dado por lo que se escogió en personalización, mientras que el estado de mover puede generar un nuevo estado de recolectar, ya que al momento de estar en movimiento puedes recoger objetos que estén en el escenario, el estado de recolectar tiene una relación con personalizar, ya que lo que recolectemos nos va a servir posteriormente para la personalización del personaje elegido, así también el estado de atacar puede desencadenar en algún momento un estado de defensa, ya que al momento de pelear no solo se debe de atacar sino que también puedes defenderte en el combate, al momento de estar en un estado de defender puede desencadenar un estado de sin puntos de vida el cual al entrar en este estado regresará a un estado de personalización para intentar de nuevo el nivel con una diferente personalización si así lo desea, por último el estado ganar está ligado con el menú de personalizar en este caso es para ver si el usuario desea volver a jugar el nivel correspondiente, este está ligado con la personalización, ya que el estado de ganar puede traer recompensas por terminar el nivel, y estas recompensas serán añadidas a sus respectivos lugares por ejemplo el inventario de poderes o de armas para poder usarlas posteriormente en otro nivel o en el mismo nivel.

* **Diagrama de actividades**

****

Cuando el usuario se decide a jugar, lo primero que se hará es cargar la partida de juego, después se inicializará el nivel, junto a sus acciones derivadas, que son inicializar el escenario, inicializar los enemigos, inicializar los personajes y cargar los artefactos. Después de que se inicializan todos los atributos de un nivel, se carga el CheckPoint de donde el jugador dejó la partida anteriormente, si el jugador no había jugado en este nivel antes, entonces el checkPoint será el inicio del nivel y del escenario. Después de cargar el CheckPoint se liberan los controles del juego y se empieza a jugar, después en el diagrama hay una condición, que hará diversas actividades, dependiendo de si el jugador ganó o no, si el jugador logró avanzar en el nivel y derrotó a todos los enemigos, entonces se imprime a pantalla su puntaje obtenido, después de guarda su proceso en la base de datos del juego, y se carga el siguiente nivel. En caso de que el jugador no ganara o que fuera derrotado, entonces se entra en otra condición que realizará el juego dependiendo de si el jugador cuenta con vidas, y si el personaje del jugador ya no tiene vidas, entonces finaliza el juego, y si aún tiene vidas, entonces solo se le resta una vida a su contador de vidas y se regresa el diagrama de actividades regresa a cargar el CheckPoint donde el jugador guardó por última vez su avance. En este diagrama de actividades, también se observa que se hace un buen uso de las clases del juego, ya que al cargar partida se inicializa el nivel, y sabemos que uno de los atributos de la clase Partida es un objeto de la clase Nivel, y también al inicializar el nivel se inicializan los atributos propios de la clase Nivel, que son el arreglo de objetos de la clase Enemigo, el Escenario, el arreglo de Artefactos y la posición del personaje.

**Conclusiones**

* Dávila Ortega Jesús Eduardo:

El entendimiento de la interacción del programa con el usuario es fundamental, ya que uno debe de saber cómo el usuario utilizara el programa y prevenir en lo posible un error o una falta de contenido , por eso es fundamental, al momento de desarrollar un software, saber la interacción que se puede llegar a tener entre clase de un mismo programa así como saber cómo se ligan entre ellas, pero a su vez es importante saber cómo se van a desencadenar los métodos y en qué orden, pero gracias a UML se puede abordar esta problemática que a veces puede resultar muy abstracta complicando en algunas situaciones el entendimiento de los procesos y el orden de ciertas acciones, pero UML brinda una guía visual la cual de cierta manera ayuda a entender cómo se supone que debería de funcionar el software, al final de cuentas considero a UML como una herramienta indispensable dentro del desarrollo de software y como este puede de alguna manera u otra facilitar la vida del programador.

* Díaz Hernández Marcos Bryan:

El modelado de los requisitos y el entendimiento de estos es algo fundamental al comienzo de los proyectos y en el desarrollo del software, debido a que el correcto análisis de estos permite marcar un camino que seguir, donde el problema se transforma a un cómo hacer, lo que permite UML, es del modelado de estos requisitos: crear un nivel, interacción con enemigos, evolución del personaje, etc. Lo que permite dar un modelo gráfico de lo que se debe hacer, además que al tener reglas y una estructura Orientada a Objetos permite modelar muy bien este paradigma. Por lo que la variedad de diagramas que se pueden llevar a cabo es complementaria y dan como resultado un gran análisis del software a realizar. Para terminar puedo decir que se cumplió el objetivo de la práctica.

* Pareja Ávila Emiliano:

Desarrollar diagramas UML puede ser algo tedioso, pero es bastante necesario al momento de hacer proyectos, ya que al tener las ideas plasmadas en un diagrama, al momento de hacer las codificaciones ya se tiene claro las interacciones de los objetos involucrados en las clases, las composiciones de dichas clases y las relaciones de herencia que pueden tener estas. Para un proyecto grande como en este caso un videojuego, los diagramas de actividades y de máquina de estados me parecieron muy importantes, porque así podemos saber cuáles son las actividades que realizará el videojuego al empezar y al terminar una acción general. Esta práctica me fue muy útil, ya que al saber cómo se hacen este tipo de diagramas, al desarrollar un proyecto que sea muy grande, podré hacer uno, desarrollando los métodos y atributos de las clases que se involucren, sus relaciones y además conocer las distintas actividades que realizará el proyecto. Por tanto los objetivos de la práctica se cumplieron.

* Vázquez Zavala Oliver Alexis:

Esta práctica me ha ayudado a reforzar diversos conceptos relacionados con el diseño e implementación de los diagramas UML, así como su importancia cuando se requiere documentar y hacer una representación gráfica de cada una de las partes que comprenden algún proyecto, en especial para programas o aplicaciones bajo el paradigma orientado a objetos, en el desarrollo de la práctica se estudiaron las características de los diversos diagramas UML que existen, tanto estáticos como dinámicos, además de en qué circunstancias se realiza e implementa cada diagrama, con el trabajo desarrollado pude entender el valor que tiene crear los diagramas UML adecuados al tipo de proyecto con el que se trabaja, ya que es una herramienta muy importante que puede ayudar a que otras personas conozcan cómo funciona, útil cuando se trabaja en equipo por ejemplo, y el objetivo de dicho programa o aplicación, además de que ayudan a tener un mejor orden sobre los pasos a seguir durante el desarrollo del proyecto.

**Bibliografía:**

* MILES, Russ, HAMILTON, Kim Learning UML 2.0 O'Reilly Media, 2006.
* Introducción a los diagramas de UML - Agile Modeling, Recuperado de: <http://www.agilemodeling.com/essays/umlDiagrams.htm>, Fecha de acceso: 10/12/2020.