Fifi Survival

David Aajenadro Sepulveda Amaya, Hector Andres Aponte , Juan Esteban Otavo Garcia, Juan Felipe Hernandez Ochoa, Julián Dario Colmenares Saenz

No. de equipo de trabajo: 01

I. INTRODUCCIÓN

En el presente documento, tenemos como objetivo explicar e informar sobre lo que es Fifi Survival y el alcance que tendrá este videojuego. Durante los diferentes apartados encontramos que problemas resuelve nuestro videojuego, su temática, a qué público se dirige nuestro videojuego y que pueden esperar de esta entrega. También establecemos los requerimientos tanto técnicos como necesarios que ha de cumplir nuestro videojuego. Como parte de la implementación de los conceptos aprendidos se definen las diferentes estructuras de datos a usar y en qué parte del proyecto estas tendrían utilidad, también encontramos aspectos tales como la muestra de un prototipo donde implementamos las estructuras de datos y su impacto en el proyecto.

II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER

El principal problema a resolver es cómo optimizar el desarrollo y experiencia de un videojuego mediante el uso de todas las estructuras de datos. Encontrando diferentes usos de estructuras y analizando en donde aplicarlas para mejorar el rendimiento del juego.

III. USUARIOS DEL PRODUCTO DE SOFTWARE

El perfil o rol que se espera del usuario, es el de una persona con gusto por los juegos. Como el producto consiste en un videojuego, se puede clasificar al usuario por el tipo de jugador y la experiencia que ofrece el videojuego.

Para clasificar el tipo de jugador se debe tener en cuenta el tiempo de juego, los cuales pueden expresar en jugadores casuales, hardcore y profesional. Para este proyecto se tendrá en cuenta el jugador casual, el cual, orienta su tiempo o uso del mismo a un pasatiempo.

Para la experiencia del videojuego, depende generalmente del gusto del usuario, en pocas palabras, el género del videojuego. Este proyecto se desarrollará en el género roguelike; es un subgénero de los RPG. Se caracterizan por ser juegos de exploración de mazmorras que se crean de manera aleatoria en cada nivel.[1]

Características de roguelike:

- Contenido aleatorio. ya sean enemigos, objetos y mazmorras.
- Para un solo jugador.
- Muerte permanente, se debe comenzar cada vez que se muera.
- Poca narrativa.
- Curva de dificultad elevada.

IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE

Como parte del proceso de análisis inicial se genera el siguiente listado de requerimientos funcionales del software, los cuales serán de vital importancia para el proceso de desarrollo del juego.

- *Visualizar inventario*

  Apertura de una lista de objetos que el personaje haya recogido en el transcurso del juego, los cuales puede elegir si los usa o los descarta.

  Opción en pantalla que el usuario al oprimir un botón seleccionado, se abrirá el inventario, al momento de volverlo a oprimir, este se cerrará.
  Al momento que el personaje se encuentre frente al objeto en el mapa, al oprimir un botón seleccionado el objeto se agregara al inventario siempre y cuando haya espacio en este, por ende, se tendrá que analizar si existe espacio en la estructura de datos del inventario para guardarlo. Si no se puede guardar, el objeto permanecerá en el piso.
  Los objetos se agregan al inventario de manera ordenada.

- *Visualizar vida del personaje*

  Barra que le indicará al usuario cuanta vida tiene su personaje.
  En todo el transcurso del juego el usuario visualizará en pantalla la vida de su personaje.

  La barra de vida iniciará al máximo, cuando el personaje reciba daño de algún enemigo, la barra de vida bajará de acuerdo al daño recibido.

- *Cantidad de enemigos en la zona*

  El usuario tendrá que interactuar con los enemigos en las diferentes zonas del juego.

  Desde el inicio de la partida el personaje deberá derrotar a una cantidad en específica de enemigos esparcidos por cada zona, para así ir desbloqueando nuevas zonas. Por cada zona desbloqueada nuestro personaje deberá derrotar aún más enemigos. Por lo que si el personaje no derrota a los enemigos necesarios no podrá avanzar al siguiente mapa.

  Cabe resaltar que por cada zona desbloqueada el mapa generado será muy distinto al generado en la zona anterior.

- *Visualizar mapa*

Espacio en pantalla en la que el usuario podrá ver de manera general la zona del mapa en la que se encuentra. En todo el transcurso del juego, el usuario podrá ver el mapa en la parte superior del juego.

Al momento que el usuario oprime un botón seleccionado, el mapa en la parte superior se agrandará, mostrando de manera más detallada la zona. Las zonas que el usuario no haya recorrido aún no se mostrarán en el mapa. Cuando el usuario vuelva a oprimir el botón, el mapa detallado desaparecerá.

- *Visualizar arma*

Espacio en pantalla en la que el usuario podrá ver el arma que su personaje utiliza.

Después de que el usuario abra el inventario, podrá seleccionar cualquiera de las armas disponibles para que inmediatamente se cierre el inventario automáticamente y el arma seleccionada esté lista para que el personaje la utilice.

- *Pausar el juego*

Opción que le permitirá al usuario pausar el juego en cualquier momento.
El usuario deberá presionar un botón en específico para que el juego se pueda pausar.

Cuando el jueo está pausado, no se podrá realizar ningún movimiento en el juego y tanto el personaje como los enemigos no podrán moverse o atacar.

- *Ingresar al juego*

Opción principal que nos permitirá inicializar nuestro juego.
Cuando el usuario seleccione la opción, se generará un mundo nuevo donde el personaje no tendrá ningún objeto

- *Salir del juego*

Opción que nos permitirá salir de nuestro juego.

El usuario al momento de pausar el juego le deberá aparecer una opción para salir del mismo, al momento de seleccionar esta opción la pestaña del juego se debe haber cerrado.

V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

Al momento de estar dentro de la partida, se tendrá una vista aérea del personaje y sus alrededores, logrando así que el usuario pueda escoger hacia dónde se quiere mover para avanzar en el juego

Vista base del videojuego

Al momento de abrir el inventario, este aparecerá en la pantalla principal, mostrando los ítems y espacios que el jugador posea en la partida

Inventario in game

VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

1) El videojuego se creará y programará mediante la plataforma Unity, la cual usa un lenguaje de programación de C#. Se podrá jugar al videojuego únicamente mediante Eun ordenador, no estará disponible para dispositivos android y ios[2].

VII. PROTOTIPO DE SOFTWARE INICIAL

En el siguiente enlace se encuentra nuestro prototipo de proyecto donde podrán ver todo lo relacionado con este videojuego.
https://github.com/znuff21/DataStructures-project

VIII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

La primera estructura de datos que se implementó corresponde al contador de enemigos. Al crear o ingresar a la partida, dentro del script NextLevelTrigger se generará una cola llamada EnemyQueue, a la cual se le encolará números enteros que va a tener n datos correspondientes a n enemigos

en el mapa. La función DesqueueEnemy desencolará un valor de la cola al momento de que el personaje mate a un enemigo. La última función es NextLevel la cual analizará si hay enemigos o no todavía en el mapa revisando la cantidad de ítems que haya en la cola. En el momento en que la cola esté vacía, se imprimirá un mensaje diciendo al jugador que puede pasar al siguiente nivel.[4]

Por cada frame se ejecutara la instancia NextLevel hasta que esta sea ejecutada una sola vez. Mientras la cola no esté vacía, es decir, existan enemigos, no se podrá avanzar al siguiente nivel.

La siguiente estructura de datos es una lista enlazada, la cual se utilizará para el inventario. Esta estructura tiene dos clases. La primera clase se llama DoubleLinkedList la cual es la implementación de una lista doblemente enlazada, la segunda clase se llama CharacterInventory la cual se encarga de la gestión del inventario. Al momento de que el personaje pase por encima de un objeto, se va a agregar a la lista enlazada del inventario y va a destruir el objeto del mapa.
Se utilizó una lista doblemente enlazada para que al momento de querer sacar algo del inventario u organizarlo, la complejidad de este disminuyera a BigO constante, logrando así que todas las acciones del inventario sea de valor constante

Al oprimir un botón seleccionado, se imprimirá en pantalla, la cantidad de objetos que el personaje lleva. Esto con el fin de que en futuros avances del proyecto, se pueda mostrar el inventario en pantalla.

IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Para todas las funcionalidades actualmente se tiene un BigO constante, debido a que aún no se ha definido el cómo usar los items, no se tiene aún una implementación para sacar items del inventario, por lo tanto no se tendrá en cuenta.[3]

Funcionalidad	Notación BigO
Encolar enemigo	O(1)
Desencolar enemigo	O(1)
Agregar item al inventario	O(1)

Desafortunadamente, al ser un videojuego, no hay manera eficaz de probar las funcionalidades con volúmenes tan grandes de datos debido a las limitaciones del motor de desarrollo Unity, pero no quita la importancia de buscar soluciones eficientes para estas funcionalidades debido a lo pesados y demandantes que pueden ser los videojuegos por todos los sistemas que están conectados e interactuando entre sí, por esto es importante que el código sea lo más eficaz posible.

X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

https://youtu.be/rVjnZk4RjTs

XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS
David Alejandro Sepúlveda Amaya	Líder	Código inicial en C# implementado en Unity
	Investigador	Aporta al plan de trabajo
	Observador	Plantilla Primera entrega
		Pruebas del prototipo
Héctor Andrés Aponte	Investigador	Consultar fuentes
		Pruebas del prototipo
	Observador	Plantilla Primera entrega
	Secretario	Facilitador de comunicación
Juan Esteban Otavo García	Experto	Código inicial en C# implementado en Unity
	Investigador	Video demostrativo del prototipo
	Técnico	Pruebas del prototipo
		Plantilla Primera entrega
Juan Felipe Hernández Ochoa	Coordinador	Programar y agendar reuniones
		Plantilla Primera entrega
	Secretario	Pruebas del prototipo
	Investigador	Consultar fuentes
Julián Darío Colmenares Sáenz	Observador	Plantilla Primera entrega
	Animador	Consultar fuentes
	Investigador	Pruebas del prototipo

XII. DIFICULTADES Y LECCIONES APRENDIDAS

La primera y más grande dificultad es aprender las tecnologías que vamos a usar. Entre estas se encuentran Git, GitHub, Unity (la cual usa como lenguaje de programación C#) y generación de mockups.

Otra dificultad fue organizar el tiempo de las reuniones y trabajos del grupo, ya que todos tenían espacios libres en horarios diferentes.

La poca experiencia en el desarrollo de videojuegos nos obligó a buscar la información necesaria para poder programarlo.

Al momento de programar el juego, fue complicado usar y enlazar los objetos del mapa con el programa para que el personaje al moverse e interactuar con el mapa, interactúa también con el código.

Aprendimos gracias a las dificultades a usar los programas que hemos estudiado, dándonos cuenta que son programas muy útiles que nos servirán desde ahora en adelante.

Generar una organización inicial nos sirvió muchísimo para poder reunirnos y avanzar en el proyecto de manera óptima, para no estar atrasados con él.

XIII. REFERENCIAS BIBLIOGRÁFICAS

[1] I. Cervera. "Roguelike | ¿Qué significa Roguelike? | Vocabuario gamer". Geekno. https://www.geekno.com/glosario/roguelike#:~:text=Un%20roguelike%20se%20trata%20de%20un%20sub-%20género,y%20se%20suelen%20realizar%20los%20movimientos%20por%20turnos. (accedido el 5 de octubre de 2022).

[2] L. Gervais, Aprender La Programación Orientada A Objetos Con El Lenguaje C#. Eni, 2016.

[3] J. T. T. Streib, Soma, Guide to Data Structures. Springer, 2017.

[4] M. T. R. D. M. Goodrich, Tamassia, Mount, Data Structures and Algorithms in C++, 2a ed. Wiley, 2004.