

Facultad de Matemáticas
Programación Estructurada
Proyecto Final. Avances del Proyecto
Docente: Emilio Gabriel Rejón Herrera



Propuesta de proyecto
Videojuego De Batallas Entre Dos Jugadores Que Utiliza Mecánicas De Lanzamiento
De Projectiles Para Obtener la Victoria (“Abbys Battle”)

C-Force

Facultad de Matemáticas UADY

Grupo D

Miércoles 08 de mayo del 2024

Integrantes de C-Force:

Alonzo Palacios Rodrigo

Cuevas García Braulio Samuel

Martínez Martínez José Pablo

Moo Pan Jareth Jaziel

Índice

<i>Índice</i>	2
<i>1.0 Antecedentes de la Propuesta</i>	5
<i>2.0 Descripción del Producto de Software</i>	10
<i>3.0 Objetivo General</i>	11
<i>4.0 Objetivos Específicos del Sistema</i>	11
<i>5.0 Diagrama de Casos de Uso</i>	12
<i>6.0 Descripción de Tipos de Usuarios</i>	13
6.1 Usuario Principal (Jugador)	13
6.1.1 Descripción	13
6.1.2 Responsabilidades	13
6.2 Administrador	13
6.2.1 Descripción	13
6.2.2 Responsabilidades	13
<i>7.0 Interfaces de Usuario</i>	14
7.1 Interfaces de menú	14
<i>8.0 Definición del estándar de codificación</i>	18
<i>9.0 Modularidad</i>	21
9.0.1 Diagrama de bloques	21
<i>9.0 Descripción de requerimientos funcionales y no funcionales</i>	22
<i>10.0 Descripción de los casos de uso</i>	28
Caso 1: Acceso al menú principal	28
Caso 2: Ingresar al juego	29
Caso 3 (Subcaso del caso 2): Tutorial	30
Caso 4 (Subcaso del caso 2): Pausar la partida	32
Caso 5: Ejecución de la partida	33
Caso 6 (Subcaso del caso 5): Colocar tropas	34
Caso 7 (Subcaso del caso 5): Turnos de juego	37
Caso 8 (Subcaso del caso 7): Disparo de proyectil	39
Caso 9 (Subcaso del caso 7): Detección de proyectiles	41
Caso 10 (Subcaso del caso 7): Determinar ganador de la partida	43

Caso 11: FeedBack	44
<i>11.0 Matriz de Requerimientos</i>	45
<i>12.0 Proceso de Desarrollo</i>	59
12.1 Herramientas utilizadas	59
<i>12.1.1 Dev C++</i>	59
<i>12.1.3 Figma</i>	59
<i>12.1.4 GitHub – GitKraken</i>	59
<i>12.1.5 WhatsApp</i>	59
<i>12.1.6 Discord</i>	59
<i>12.1.7 Canva</i>	60
12.2 Organización	60
<i>12.2.1 WhatsApp – Discord</i>	60
<i>12.2.2 To do List</i>	60
12.3 Monitoreo	61
<i>12.3.1 GitHub – GitKraken</i>	61
12.4 Bitácoras	61
<i>12.4.1 Microsoft To Do</i>	61
12.5 Medición del trabajo grupal	62
12.6 Medición del trabajo individual	63
<i>13.0 Primer avance de código</i>	65
<i>14.0 Reporte final del código</i>	72
<i>Funcionalidad de los requerimientos implementados</i>	72
RF-001: Menú principal.	72
RF-006: Salir del sistema.	72
RF-005: Ingresar al juego.	74
RF-007: Tutorial.	74
RF-012: Interacción multijugador local.	77
RF-013: Tablero de juego.	77
RF-014: Colocar tropas.	78
RF-016: Trayectoria de proyectiles.	81
RF-017: Turnos de jugadores.	84
RF-018: Daño de área.	84
RF-019: Detección de proyectiles.	85
RF-020: Determinar ganador.	85
RF-021: FeedBack.	85

RF-022: Pausar la partida.	86
Acceso al código completo	87
<i>15.0 Reporte de evaluación basada en objetivos</i>	<i>88</i>
<i>Referencias</i>	<i>90</i>

1.0 Antecedentes de la Propuesta

Al realizar este proyecto de software se consideró que en este mundo ya se han explorado un sin número de ideas, por lo que se puso como objetivo tomar una idea y reinventarla para funcionar en el lenguaje de programación C. En base a lo anterior se realizó una lluvia de ideas para encontrar que tipo de proyecto se realizaría, al final se optó por la realización de un videojuego como el producto de software.

Al escoger el tema que se planeaba realizar, surgió una sencilla, aunque no menos interesante problemática. Es común que cuando un grupo de dos o más amigos no cuentan con acceso a internet en donde se ubican, sea su casa o algún lugar donde se encuentren de invitados, entre otros, suelen entrar a pie los juegos de mesa para matar el tiempo, pero si no hay suficiente espacio para jugar de forma física, o incluso si no cuentan con ningún juego de mesa físico, una alternativa serían juegos sencillos como “Gato” o “Conecta 4” que se pueden jugar con papel o un lápiz, sin embargo, actualmente muchas personas ya cuentan con dispositivos electrónicos como pueden ser celulares, por ello este proyecto pretende tomar la problemática anterior y proporcionar un videojuego que sea divertido para los usuarios.

El reto de realizar este proyecto de software sería realizarlo en lenguaje C, ya que es estructurado y no orientado a objetos como lo es C++. El producto de software será un juego de video para dos personas, que consiste en un enfrentamiento 1v1 en un tablero designado por casillas, con bases en el popular juego “Battle ship”. Para este se decidió utilizar una temática “Medieval” para el diseño de “Personajes” y “Ambiente”.

Por último, ya que el producto está basado en el videojuego “Battle Ship” es evidente que aquellos softwares que sean un videojuego con temática de batallas navales son nuestras principales competencias, por ello es importante resaltar cuáles son las características de estos.

En internet existen un sin fin de videojuegos que tienen temática de batallas navales, estos están alojados en páginas web o cómo un juego gratuito para celular o para computadoras. Se considera que los dos principales son los que se presentan a continuación.

El “Battle Ship” de la aplicación de juegos llamada como “Plato” que se encuentra disponible para celulares y para computadoras, es un software que debido a sus características posee similitudes con el nuestro. Sus características principales son las siguientes:

- Es un videojuego para dos personas, jugándose por turnos.
- Se juega online, desde dispositivos diferentes.
- El tablero de juego es de 10x10 casillas, contando cada jugador con 6 navíos.
- En su respectivo turno cada jugador coloca en el tablero sus barcos al empezar la partida.
- En sus turnos los jugadores pueden elegir de forma directa y sin dificultad la casilla en donde caerá el proyectil (este no último no es lanzado, solo representa en que casilla busca atacar el usuario), además los jugadores no ven donde coloco su rival a sus embarcaciones, radicando la dificultad en descubrir donde están las naves enemigas.
- El juego termina cuando uno de los dos jugadores elimina todas las embarcaciones del rival.
- El videojuego fue realizado principalmente en el lenguaje de programación conocido como “JavaScript”.

- Es un videojuego alojado dentro de un software más complejo que aloja muchos otros juegos.

Las diferencias que presenta este software con este proyecto son las siguientes:

- El lenguaje utilizado para desarrollar este software es estructurado, siendo conocido como C.
- “Abyss Battle” se juega de forma local, es decir, desde el mismo dispositivo y sin necesidad de tener acceso a internet.
- En el tablero se colocan 12 soldados, siendo así el doble de elementos que en “Battle Ship” de la aplicación “Plato”.
- Este proyecto no se encuentra almacenado en otro software. En su lugar, “Abyss Battle” cuenta con un menú principal para navegar entre sus opciones.
- Para disparar un proyectil en “Abyss Battle” no se selecciona la casilla donde se desea lanzar, en su lugar los jugadores tienen que buscar darles a los soldados del contrincante al lanzar su proyectil por medio de un cañón, ya que el lanzamiento depende de una trayectoria acertar el tiro no es seguro. Los jugadores pueden ver donde colocaron cada una de sus tropas mutuamente a diferencia de “Battle Ship”.
- El tablero de juego cuenta con dimensiones de 20x90 casillas. Siendo 20 filas y 90 columnas.

Por otro lado, tenemos al software llamado “Sea Battle 2”, este es un videojuego desarrollado para dispositivos móviles (celulares), es gratuito, pero cuenta con pagos dentro de la aplicación. Sus principales características son las siguientes:

- Se desarrollo usando un lenguaje de programación orientado a objetos.

- Fue desarrollado para dispositivos móviles, por ello no se puede jugar en computadoras (a menos de que se use un programa externo cómo puede ser un emulador).
- Su principal modo de juego es el “Online”, es decir que puedes jugar contra otros jugadores de todo el mundo.
- Cuenta con un modo juego “offline”, este es contra un "bot" y sirve de entrenamiento. Además, tiene un modo local para jugar por turnos contra tus amigos.
- Puedes diseñar una ciudad portuaria en el juego, esto es una funcionalidad adicional a las partidas principales.
- Existe una clasificación en el modo “online”, en donde incluso se llevan a cabo torneos en donde puedes o no participar.
- Los jugadores pueden chatear entre ellos durante la partida usando emojis.
- El tablero de juego es de 10 x 10 casillas, teniendo 10 filas y 10 columnas.

Las principales diferencias entre “Sea Battle 2” y este proyecto son las presentadas a continuación:

- “Abyss Battle” es un videojuego desarrollado en un lenguaje estructurado conocido como C.
- Nuestro proyecto solo cuenta con un modo local para jugar entre dos usuarios por turnos.
- El modo de eliminar a las tropas enemigas en “Abyss Battle” es por medio del lanzamiento de un proyectil por medio de un cañón, a diferencia de “Sea Battle 2” donde se usan poderes para atacar casillas seleccionadas donde puede o no que estén las embarcaciones enemigas.
- “Abyss Battle” es un proyecto pensado para su ejecución en computadoras, no para dispositivos móviles.

Con la información anteriormente expuesta, se puede concluir que la principal diferencia que presenta el proyecto “Abyss Battle” con respecto a software similares es la mecánica de lanzamiento de proyectiles por medio de cañones para eliminar tropas enemigas, siendo esta algo que no se encontró en ningún otro videojuego con temática de batallas navales.

2.0 Descripción del Producto de Software

Nombre del Producto de Software: Videojuego De Batallas Entre Dos Jugadores Que Utiliza Mecánicas De Lanzamiento De proyectiles Para Obtener la Victoria (“Abbys Battle”).

El producto de software lleva por nombre “**Abyss Battle**” elaborado en lenguaje C. Será un videojuego para dos jugadores, de formato local y para ordenadores de sobre mesa y portátiles de bajos recursos, considerando cómo bajo recursos a aquellos ordenadores con a lo menos 4GB de memoria RAM, que cuenten con sistema operativo “Windows”. El proyecto estará ambientado en la época medieval en sus personajes, escenarios y mecánicas, además carecerá de una historia y tendrá como principal objetivo desarrollar e implementar una mecánica de lanzamiento de proyectiles.

En “**Abyss Battle**” al iniciar una partida los usuarios eligen donde colocar a sus soldados, permitiendo que los jugadores puedan ver donde colocó a sus soldados el contrincante. Para eliminar tropas del contrincante se usará un cañón que tendrá una mecánica de tipo lanzamiento cargado, esta consiste en que el jugador podrá dirigir y cargar un proyectil, en la pantalla se mostrará solo una parte de la trayectoria que seguirá el proyectil, teniendo como meta principal impactar dicho proyectil sobre una de las tropas enemigas. Obtiene la victoria aquel jugador que acabe con todos los soldados del contrincante.

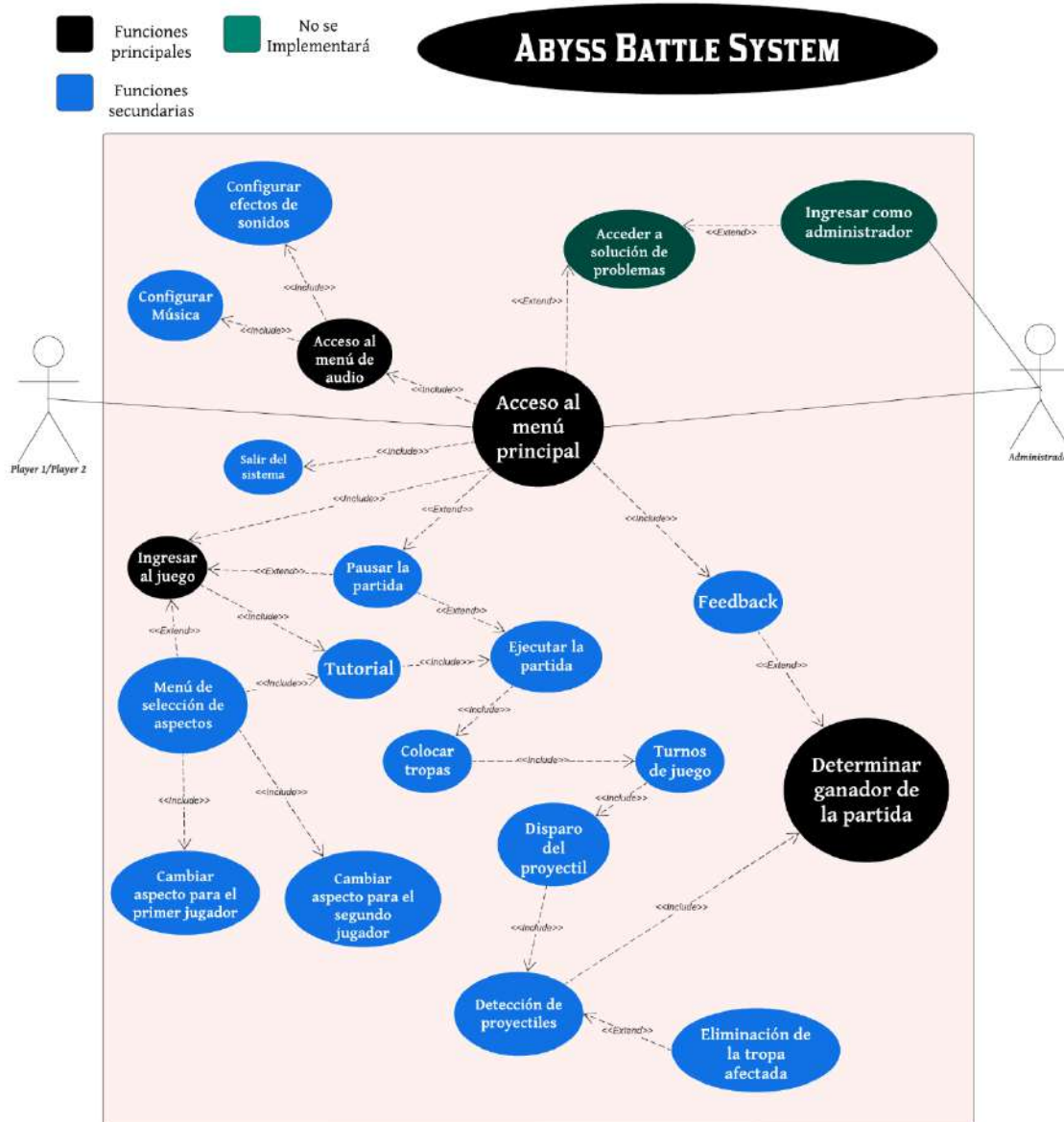
3.0 Objetivo General

Desarrollar videojuego en formato 2D, que simule un combate estratégico por turnos entre dos jugadores, implementando una mecánica de lanzamiento y precisión con un cañón para su ejecución en consola de sistema “Windows”.

4.0 Objetivos Específicos del Sistema

1. Programar una mecánica que permita otorgar una potencia de lanzamiento, direccionar y calcular la trayectoria de un proyectil cuando es disparado.
2. Optimizar el software para garantizar un rendimiento fluido en computadoras de escritorio y portátiles que cuenten con sistema operativo “Windows”.
3. Implementar un sistema de juego basado en turnos, que además permita detectar cuando ocurre un cambio de turno.
4. Crear interfaces minimalistas e intuitivas para el sistema, que permitan que los usuarios se adapten y aprendan a utilizar las mecánicas del videojuego.
5. Codificar un algoritmo para detectar las colisiones entre un proyectil y otros elementos del software.
6. Incorporar en el software gráficos en 2D, animaciones, efectos de sonido y un “soundtrack” de uso libre.

5.0 Diagrama de Casos de Uso



6.0 Descripción de Tipos de Usuarios

6.1 Usuario Principal (Jugador)

6.1.1 Descripción: El jugador es la persona que interactúa directamente con el software, ya sea controlando el cañón y lanzando proyectiles durante las partidas o configurando opciones dentro del juego.

6.1.2 Responsabilidades:

- Jugar partidas contra otro jugador.
- Ganar partidas contra otros jugadores
- Configurar opciones del juego, como ajustes de audio.
- Explorar y descubrir las mecánicas del juego

6.2 Administrador

6.2.1 Descripción: El administrador es responsable de gestionar aspectos técnicos y de gestión del juego.

6.2.2 Responsabilidades:

- Realizar actualizaciones y mantenimiento del juego.
- Solucionar problemas técnicos o de rendimiento que presenten los usuarios principales.

7.0 Interfaces de Usuario

7.1 Interfaces de menú



Figura 1: Interfaces de los menús del sistema.

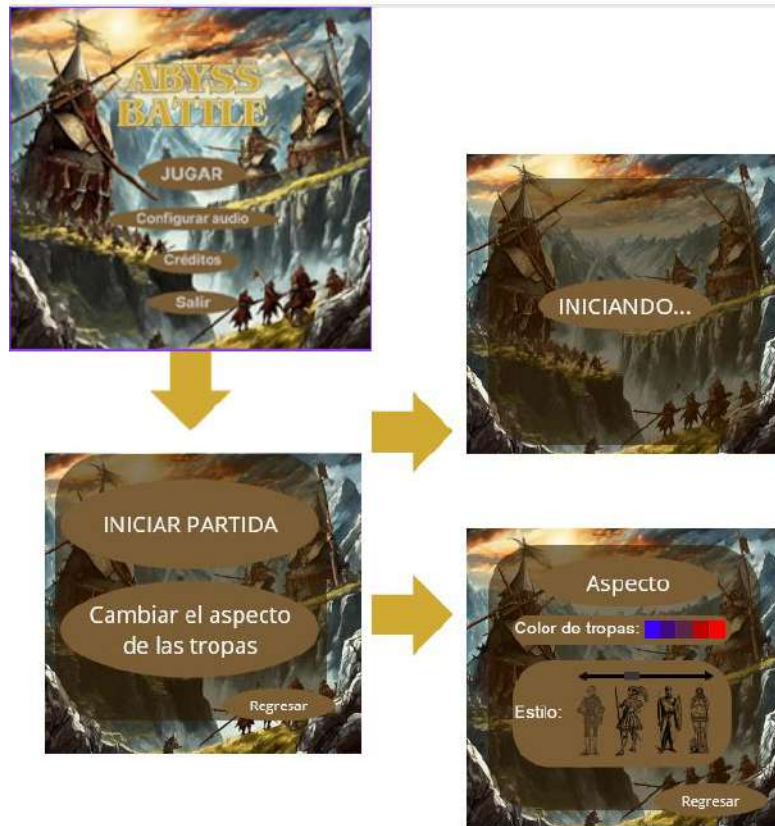


Figura 2: Flujo de las interfaces que el usuario visualiza al iniciar una partida

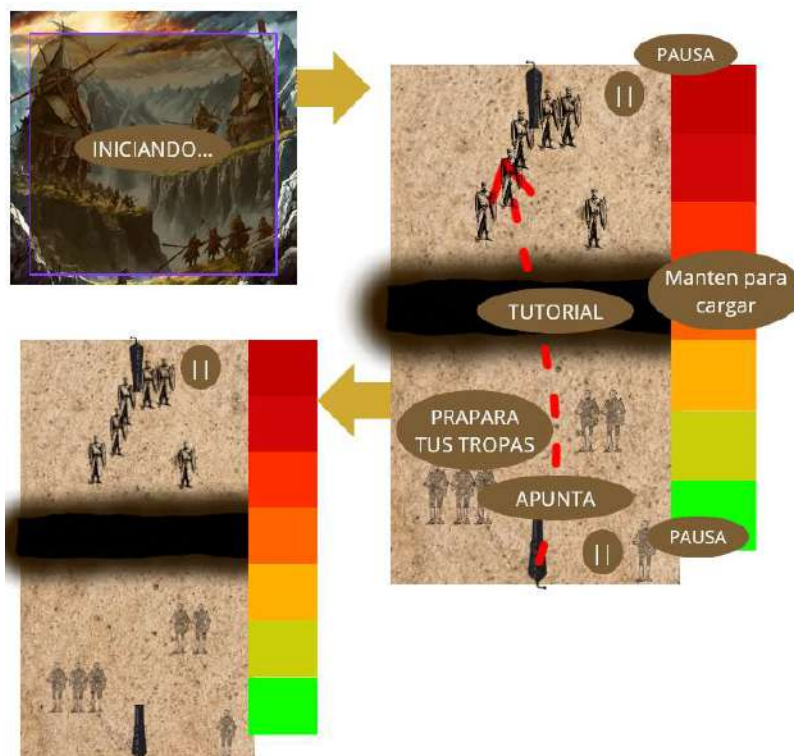


Figura 3: Interfaces que visualiza el usuario durante la ejecución de una partida normal en le software.

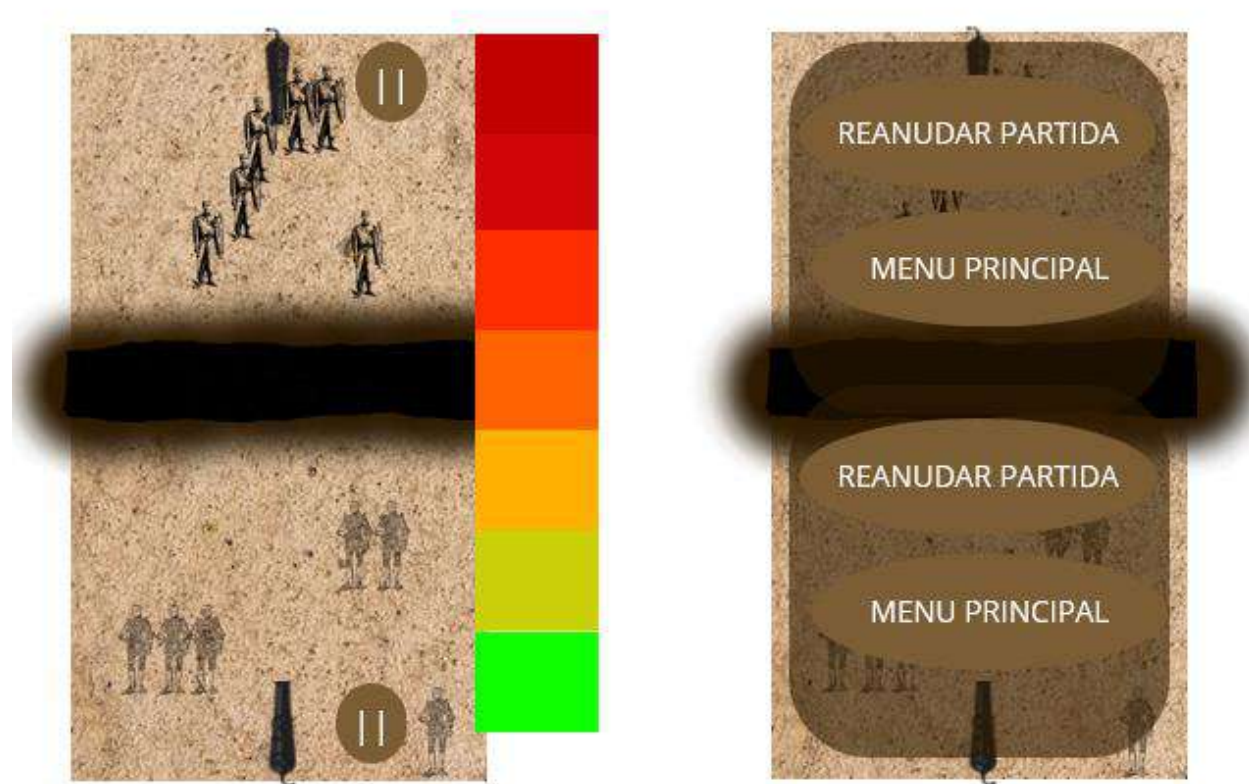


Figura 4: Interfaz del menú de pausa durante la ejecución de una partida.

8.0 Definición del estándar de codificación

De forma general se utilizará el compilador de “Dev C++” en su versión 5.11 para el desarrollo de la lógica del producto de software, por otro lado, se usará la biblioteca externa freeglut.h para habilitar funciones de OPenGL y poder añadir ventanas y gráficos.

para generar los gráficos del videojuego.

Para desarrollar un proyecto de este tamaño, será necesario llevar una documentación adecuada de los archivos importantes para su elaboración, además se debe establecer un estándar de codificación para promover el mutuo entendimiento entre integrantes y así optimizar el tiempo de codificación, ya que de no realizarse se extenderá de forma innecesaria el desarrollo del producto de software.

Se consideró una alternativa viable para establecer una forma legible de realizar el código, al "Estándar de Codificación de GNU", un proyecto de software libre iniciado en enero de 1984 a cargo de Richard Stallman, por ello cuenta con mucha documentación sobre el mismo. GNU consiste en un amplio conjunto de programas, con aplicaciones, librerías, herramientas para el desarrollo y juegos; con el tiempo se volvió tan grande que se tuvo que implementar reglas y recomendaciones para mantener un código legible y de alta calidad.

Con base al estándar anteriormente mencionado se busca establecer pautas claras y específicas para el proyecto "Abyss Battle", tomando los aspectos que se consideran más relevantes de GNU. A continuación, se proporciona el estándar que se usará para realizar este proyecto de software.

Estándar de Codificación para el Proyecto "Abyss Battle"

1. Legibilidad y Claridad:

- El código debe ser fácil de leer y entender para cualquier persona que lo revise.
- Utiliza nombres descriptivos y significativos para las variables, funciones y archivos.

2. Consistencia:

- Sigue un estilo de codificación consistente en todo el proyecto.
- Utiliza el mismo estilo de nombrado de variables, funciones y archivos en todo el código.
- Mantén una estructura de directorios consistente para los archivos del proyecto.

3. Portabilidad:

- Escribe código que sea portable y pueda ejecutarse en diferentes sistemas y compiladores.
- Evitar características específicas de una plataforma o compilador en tu código, salvo que sea necesario.

4. Uso de Estándares Aceptados:

- Sigue las convenciones y estándares de la industria para el nombrado de variables, funciones y archivos.
- Utiliza las bibliotecas estándar de C y SDL de manera apropiada y conforme a sus respectivas convenciones de codificación.

5. Documentación y Comentarios:

- Documenta el propósito y funcionamiento de cada función y módulo en comentarios claros y concisos.

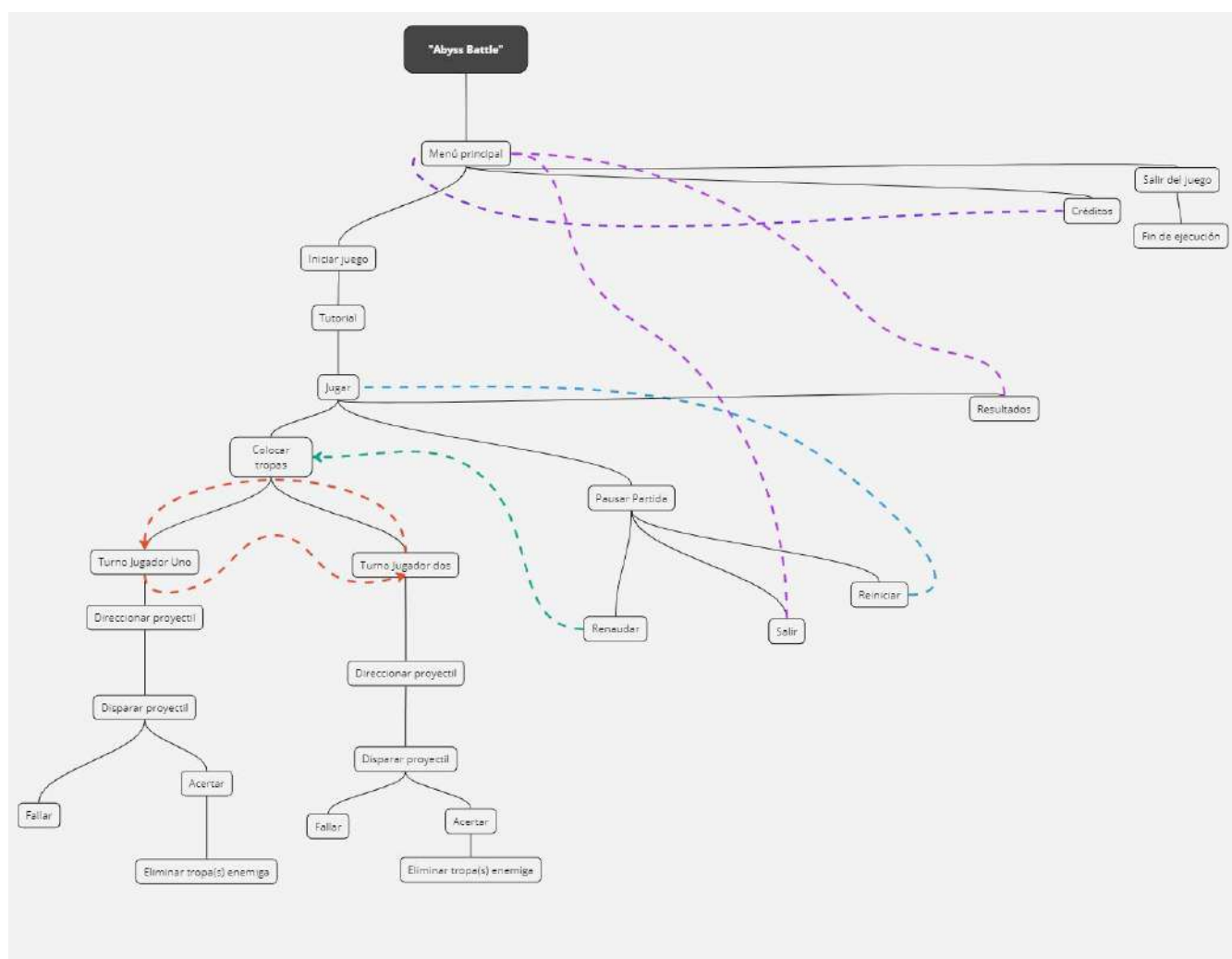
- Utiliza comentarios para explicar partes importantes del código, incluyendo algoritmos complejos, decisiones de diseño y aspectos no obvios.

El "Estándar de Codificación de GNU" no es un documento único, sino más bien un conjunto de convenciones y prácticas de codificación promovidas por el Proyecto GNU. Revisar referencias relacionadas al final del documento.

9.0 Modularidad

9.0.1 Diagrama de bloques

La siguiente imagen es el diagrama de bloques del sistema “**Abyss Battle**”, este explica la relación que existe entre los diferentes módulos del videojuego.



9.0 Descripción de requerimientos funcionales y no funcionales

Requerimientos funcionales (RF) - Tabla 1	
Nombre	Descripción
RF-001. Menú principal	El sistema deberá contar con un menú principal de opciones que se le muestre en pantalla al usuario al iniciar la ejecución del software. Dicho menú tendrá las siguientes opciones: <ol style="list-style-type: none"> 1. Jugar 2. Configurar audio 3. Créditos 4. Salir
RF-002. Menú de audio	Eligiendo el botón con la opción de “Configurar audio” en el menú principal, el sistema permite a los usuarios ingresar al “menú de audio”, un apartado en donde se verá la barra de nivel de volumen, este servirá para ajustar las configuraciones de sonido del juego.
RF-003. Configurar música	En el “menú de audio” existe una barra lateral que permite subir, bajar o silenciar por completo el volumen de la música del sistema, este está barra lateral se tendrá un círculo para elegir el nivel de volumen que se desee; si dicho círculo se desplaza hacia la izquierda, se bajará el nivel del volumen, de lo contrario, se aumentará el nivel del volumen.
RF-004 Configurar efectos de sonido	En el “menú de audio” existe una barra lateral que permite subir, bajar o silenciar por completo el volumen de los efectos de sonido del sistema, este está barra lateral se tendrá un círculo para elegir el nivel de volumen que se desee; si dicho círculo se desplaza hacia la izquierda, se bajará el nivel del volumen, de lo contrario, se aumentará el nivel del volumen.
RF-005. Ingresar al juego	Los usuarios podrán ingresar a la ejecución de una partida del juego usando la opción de “jugar” en el menú principal, también podrían ingresar a la ejecución de la partida presionando la tecla “Enter” de su ordenador de mesa o portátil en el mismo menú. El sistema redirige a los usuarios a la pantalla de juego iniciando una partida directa.
RF-006. Salir del sistema	El sistema permite que el usuario pueda salir de este último usando el botón de “salir” ubicado en el menú principal, al presionarlo automáticamente se cerrará el juego, dejando de ejecutarlo.

RF-007. Tutorial	<p>Al iniciar la ejecución de la partida mediante el botón “jugar”, independientemente de si se eligieron aspectos o no para los personajes, el software despliega en pantalla del usuario un tutorial que le explica cómo funcionan las principales mecánicas del juego:</p> <ul style="list-style-type: none"> • Al seleccionar las tropas <ul style="list-style-type: none"> ○ Flecha arriba / W: Desplazar la tropa un espacio arriba ○ Flecha abajo / S: Desplazar la tropa un espacio abajo ○ Flecha derecha / D: Desplazar la tropa un espacio a la derecha ○ Flecha izquierda / A: Desplazar la tropa un espacio a la izquierda ○ Enter: Colorar la tropa en el espacio situado • Al direccionar el cañón <ul style="list-style-type: none"> ○ Flecha arriba / W: Direccionar el cañón hacia arriba ○ Flecha abajo / S: Direccionar el cañón hacia abajo ○ Enter: Seleccionar posición del cañón • Al potenciar el cañón <ul style="list-style-type: none"> ○ Flecha derecha / D: Incrementar la potencia ○ Flecha izquierda / A: Decrementar la potencia ○ Enter: Seleccionar la potencia del cañón
RF-008. Objetivo de destrucción	<p>Durante la ejecución del tutorial se deberá establecer al o los usuarios que como objetivo principal del juego se debe buscar la destrucción de objetos, es decir, la eliminación de objetivos específicos (las tropas enemigas) mediante los lanzamientos de proyectiles, mencionando que el objetivo busca derribar todas las tropas enemigas para ganar.</p>
RF-009. Menú de selección de aspecto.	<p>Después de seleccionar la opción “jugar” en el menú principal, pero antes de iniciar una partida, el sistema mostrará en pantalla una ventana que pregunta si se desea ingresar al menú de selección de aspecto. Si el usuario selecciona la opción “sí” se mostrará en pantalla dicho menú para que los jugadores pueden elegir el aspecto físico que tendrán sus tropas en el tablero de juego. En dicho menú estarán las siguientes opciones:</p> <ol style="list-style-type: none"> 1. Cambiar aspectos del jugador 1. 2. Cambiar aspectos del jugador 2. 3. Aceptar. <p>Al elegir la opción de “Aceptar” el menú se cerrará y se terminará la selección de aspecto para sus personajes.</p>
RF-010. Detectar la selección de aspectos.	<p>Cuando se selecciona un aspecto por parte del usuario el sistema detecta cuál de los jugadores ha seleccionado el aspecto de sus tropas, deshabilitando este aspecto para el jugador contrario al que selecciono.</p>

RF-011. Cambiar el aspecto para el primer o segundo jugador	Se selecciona la opción “Cambiar aspectos del jugador 1” o en su defecto “Cambiar aspectos del jugador 2”. El software despliega en pantalla para los usuarios una galería con todos los aspectos disponibles, el usuario puede desplazar de izquierda a derecha en esta galería dando “click” con el cursor del “ratón” a unos botones ubicados a los lados de la pantalla (estos tendrán flechas indicando la dirección a la que mueven la galería). Cuando tenga seleccionado el aspecto que desee y al darle con el cursor a la opción de “aceptar” centrada en la parte inferior, se habrá completado el cambio de aspecto y se regresará al “menú de selección de aspecto”.
RF-012. Interacción multijugador local.	Al iniciar la ejecución de una partida del juego, es decir, ya que se haya seleccionado la opción de “jugar” y se haya mostrado el tutorial, el software permite a los jugadores interactuar entre ellos en un entorno multijugador local de 1v1, asignándole turnos de acción al jugador uno y al jugador dos, de forma que el usuario que maneja el sistema se convertirá en un turno en el primer jugador, y en el siguiente será el segundo jugador, evitando que ambos bandos interactúen en el mismo momento.
RF-013. Tablero de juego	El software dentro de la escena designada para la ejecución de las partidas (donde los usuarios pueden jugar) se cuenta con un tablero de juego de 20 casillas en forma de filas con 90 casillas en forma de columnas, este tablero se le mostrará en la pantalla al usuario. Deberá dividirse a la mitad por una franja de casillas donde ninguno de los dos jugadores tendrá permitido el despliegue de tropas, esta franja inhábil de casillas estará designada para ser el abismo que divide el tablero, el cual da nombre al juego.
RF-014. Colocar tropas	Los usuarios durante la ejecución de la partida pueden elegir donde colocar a sus 12 soldados en las casillas validas del tablero de juego. Esta acción se realizará al iniciar la partida, empezando con el jugador uno, en el que tendrá que colocar todas sus tropas a su elección para que el siguiente jugador realice la misma acción.
RF-015. Potencia de proyectiles	Se implementa una mecánica, durante la ejecución de una partida de juego, que permita cargar la potencia de un proyectil al elegir dicha potencia en un intervalo del 1 al 11, siendo el 1 la potencia más baja y el 11 la potencia más alta. Se podrá elegir utilizando las flechas izquierda y derecha, o los botones “A” y “D” para cambiar el valor, se establecerá la potencia elegida por el usuario dándole “click” al botón “Enter”.
RF-016. Trayectoria de proyectiles	Se calcula de forma interna la trayectoria que llevará el proyectil que será lanzado, mostrando en la pantalla para el jugador en turno el inicio de esta trayectoria previamente calculada, evitando que se le muestre el total de la trayectoria al usuario, la trayectoria que se verá será del lado del jugador en turno, mostrando con flechas hacia donde se dirigirá el proyectil, se puede cambiar la

	trayectoria hacia arriba o abajo utilizando los controles “W” / Flecha arriba o “S” / Flecha abajo respectivamente, se establecerá la trayectoria con el botón “Enter”.
RF-017. Turnos de los jugadores	Según los jugadores colocan a cada soldado, el software deberá mostrar en pantalla cuántos le faltan por colocar, así como detectar cuándo termine de colocar a todas sus tropas. Al terminar el tiempo designado, el sistema deberá cambiar el turno y actuar al segundo jugador, quien podrá colocar sus tropas igual que el primero. El software deberá ser capaz de reconocer cuando ambos jugadores hayan terminado de colocar sus tropas, así iniciando la ejecución normal del cambio de turnos, cambiando de turno del jugador uno al dos y viceversa cada vez que ellos realicen una acción, es decir, disparen un proyectil con su cañón, pero si cualquiera de los dos jugadores realiza un tiro exitoso en su cañón se le permitirá realizar otro tiro, así hasta que falle, al fallar automáticamente se cambiará de turno al jugador rival.
RF-018. Daño de área	Los proyectiles lanzados por el cañón de cualquiera de los usuarios no solo pueden eliminar una tropa enemiga por contacto directo, si no que deberán contar con daño de área para favorecer la inmersión en el juego. El daño de área podrá eliminar tropas enemigas que estén alrededor de donde impacto el proyectil que haya sido lanzado, en caso de no haber tropas enemigas alrededor, solo se eliminará a la que haya impactado directamente, o en su defecto, no se eliminará a ningún soldado enemigo en caso de fallar el disparo.
RF-019. Detección de proyectiles	El software detecta cuando un proyectil impacta directa o indirectamente (por daño de área) sobre una tropa enemiga durante la ejecución del turno de uno de los jugadores, eliminando el Sprite de dicha tropa de la pantalla, y reduciendo el número de tropas del jugador que perdió a su guerrero en uno. Al detectarse un impacto de un proyectil contra una tropa enemiga, el sistema añade permite realizar otro tiro en el turno. Si no sucede impacto, el tiempo del turno se acaba automáticamente y se pasa al siguiente turno. Al eliminar la tropa enemiga, automáticamente aparecerá una “X” marcada en el espacio donde se sitúa la tropa eliminada.
RF-020. Determinar ganador	Después del lanzamiento de cualquiera de los jugadores y de que el sistema haya detectado la colisión o no colisión del proyectil, también analiza y determina cuando uno de los dos jugadores se queda sin tropas en el tablero, en caso de que se cumpla esta condición entonces la partida termina en ese momento. Al terminarse la partida en pantalla se despliega cuál de los dos usuarios fue el ganador y se devuelve a la pantalla de menú principal.

RF-021. FeedBack	Al terminar la partida, se retroalimentará a los dos jugadores sobre el número de lanzamientos, derribos y tropas restantes que tuvo cada uno. Esta retroalimentación se imprimirá en pantalla al usuario en forma de ventana y se deberá seleccionar la opción de “Aceptar” centrada en la parte inferior para terminar de forma oficial la ejecución de una partida del juego.
RF-022. Pausar la partida	El software cuenta con un botón de pausa durante la escena donde se desarrolla la partida entre los jugadores, deberá ubicarse en la esquina superior izquierda de la pantalla y permite desplegar una ventana opciones. Entre las opciones se encuentra “salir al menú principal” o reanudar la partida. Al seleccionar “Salir al menú principal” el sistema devuelve al usuario al menú principal.
RF-023. Solución de problemas	Se cuenta con una opción en el menú principal para contactar a los administradores para la solución de problemas bien para ingresar como un administrador. Si no es administrador, se deberá seleccionar la opción de ayuda y luego se desplegará en pantalla, para los usuarios, la información de contacto. Para esta opción será necesario que exista una conexión a internet.
RF-024. Ingresar como administrador	Los administradores pueden ingresar al sistema para favorecer a la resolución de problemas de los usuarios. Para ello necesitarán contar con conexión a internet, ingresar a la opción administradores del menú principal, posteriormente seleccionar “soy administrador” e ingresar su usuario y contraseña.

Nota: Los colores que se encuentran en la tabla 1 y tabla 2 están basados en el método MoSCoW. Estos explican la importancia de cada uno de los requerimientos basándose en los resultados de la matriz de requerimientos y después en el criterio del equipo. Color verde representa aquellos que consideramos esenciales para el sistema. Color amarillo representa los requerimientos que deberían agregarse para mejorar la calidad, pero no son esenciales. Azul aquellos que pueden agregarse sin mucha dificultad. Rojo aquellos que fueron descartados por la matriz de requerimientos ya que no concuerdan con nuestros objetivos.

Requerimientos no funcionales (RNF) - Tabla 2	
Nombre	Descripción
RNF-001	El sistema debe estar optimizado para poder utilizarse en computadores que cuenten con un sistema operativo Windows 8 en adelante.
RNF-002	El juego debe ser capaz de ejecutarse de manera fluida y sin retrasos en el dispositivo, manteniendo una tasa de “frames” por segundo (FPS) adecuada.
RNF-003	Se debe optimizar para que funcione de forma fluida en computadoras con mínimo 4 GB de RAM en adelante.
RNF-004	El software debe contar con un diseño de interfaces intuitivas y minimalista que permita a los usuarios navegar fácilmente entre sus opciones y que además ayude a tener un rápido aprendizaje de cómo utilizarlo.
RNF-005	Los controles del juego responden de manera precisa y segura a las acciones del jugador, reaccionando a las acciones que éste decida
RNF-006	Debe estar optimizado para su ejecución de forma local en ordenadores de sobre mesa y portátiles (laptops), es decir, no se debe requerir que exista una conexión a internet para poder utilizar el sistema.
RNF-007	El sistema implementa gráficos y animaciones retro en 2D para que los jugadores tengan mayor inmersión, con el objetivo de reforzar la temática medieval del juego.
RNF-008	Cuenta con efectos de sonidos y música ambiental (“soundtrack”). La música ambiental debe cambiar por cada escenario, es decir, será diferente para los menús con respecto a la ejecución de la partida.

10.0 Descripción de los casos de uso

Caso 1: Acceso al menú principal

Descripción del caso de uso

Actor(es)	Jugador 1 - 2, Administrador.
Descripción	Tanto el usuario que funge como los jugadores como el administrador al ejecutar el software se les mostrará en pantalla el menú principal.
Precondiciones	<ul style="list-style-type: none"> El sistema debe estar instalado en un dispositivo que cuente con el sistema operativo Windows. El usuario debe ejecutar el sistema en su computadora.
Poscondiciones	El usuario visualiza el menú principal en su pantalla.
Subcaso(s)	Ninguno

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	El usuario ejecuta el sistema en su computadora	1	<p>La aplicación muestra en la pantalla del usuario el menú principal del sistema, mientras inicia la ejecución del sonido de la aplicación.</p> <p>El menú principal despliega las siguientes opciones:</p> <ol style="list-style-type: none"> 1. Jugar 2. Configurar audio 3. Créditos 3. Salir 	

Flujo de caso de uso

Excepciones: Este caso de uso no cuenta con ninguna excepción

Caso 2: Ingresar al juego

Descripción del caso de uso

Actor(es)		Jugador 1 - 2
Descripción	El usuario que funge como los jugadores. Ingresa a la ejecución de una partida del juego utilizando como medio el menú principal.	
Precondiciones	<ul style="list-style-type: none"> • Debe haber ejecutado previamente la aplicación en su computadora. • Debe haberle dado “click” a la opción de “jugar” en el menú principal. 	
Poscondiciones	El sistema despliega el tutorial de del juego en la pantalla.	
Subcaso(s)	Tutorial, pausar partida.	

Actor			Sistema	
Paso	Acción	Paso	Acción	Excepción
1	El usuario selecciona la opción de “jugar” en el menú principal.	1	El sistema cambia de pantalla y muestra ahora al usuario las siguientes opciones: 1. Tutorial 2. Iniciar partida	
2	Se selecciona “Iniciar partida” en el menú desplegado previamente.	2	El sistema empieza a cargar, una vez termine de cargar el escenario despliega en pantalla en tablero de juego.	E1

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	Se selecciona la opción de “tutorial”.	El sistema se dirige al subcaso “Tutorial”.

Caso 3 (Subcaso del caso 2): Tutorial

Descripción del caso de uso

Actor(es)		Jugador 1 - 2
Descripción	El sistema despliega en la pantalla del usuario una ventana dando la bienvenida al tutorial. Tiene como objetivo explicar a los jugadores las principales mecánicas del software durante una partida.	
Precondiciones	<ul style="list-style-type: none"> • Debe haber seleccionado la opción de jugar en el menú principal. • Debe haber seleccionado la opción de iniciar partida o en su defecto haber pasado por la selección de aspecto y después haberle dado a iniciar. 	
Poscondiciones	Se desea suerte a los jugadores e inicia la partida.	
Subcaso	Ninguno.	

Actor			Sistema	
Paso	Acción	Paso	Acción	Excepción
1	El usuario inicia la partida	1	En el sistema se despliega una ventana dándole la bienvenida, preguntando si desea ver el tutorial.	
2	Se le indica al sistema que sí se desea ver el tutorial.	2	Se despliega una pantalla de información con una explicación básica sobre el funcionamiento de las mecánicas del juego. Indicándole al usuario que para cerrar el tutorial se debe darle presionar el botón de “enter”.	E1
3	Se presiona el botón de “enter”.	3	Se termina el y se cierra la ventana de información.	

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	Se le indica al sistema que se desea “iniciar la partida”.	El sistema ejecuta directamente el paso 2 del caso 2.

Caso 4 (Subcaso del caso 2): Pausar la partida

Descripción del caso de uso

Actor(es)	Jugador 1 - 2, Administrador.
Descripción	Los jugadores presionan el botón de “Esc” durante la ejecución de la partida.
Precondiciones	<ul style="list-style-type: none"> Se debe haber iniciado una partida en el sistema.
Poscondiciones	La partida se detiene, ahora los usuarios pueden elegir entre tres opciones.
Subcaso	ninguno

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	Durante la ejecución de la partida y en cualquiera de los turnos de los jugadores, se presiona el botón de “Esc” en el durante la ejecución del juego.	1	El sistema despliega en pantalla una ventana con dos opciones: 1. Reanudar partida 2. Reiniciar partida 3. Salir del juego	E1
2	El usuario selecciona “Regresar al menú principal”	2	El programa devuelve al menú principal, imprimiéndolo nuevamente en pantalla.	E2, E3

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	El usuario selecciona la opción de “reanudar partida”.	Se devuelve a la pantalla de la partida justo donde se quedó.
E2	El usuario selecciona la opción de “Reiniciar partida”	La partida se reinicia, empezando por colocar nuevamente las tropas (Caso 6)

Caso 5: Ejecución de la partida

Descripción del caso de uso

Actor(es)		Jugador 1 – 2.
Descripción		Se imprime en pantalla el tablero de juego vacío en donde se encuentra un cursor controlado con las teclas “w”, “a”, “s” y “d”. Indicando el jugador que se encuentra en turno.
Precondiciones		<ul style="list-style-type: none"> • Se debió haber elegido en el menú principal la opción de “Jugar” • El usuario debió haber elegido ver el tutorial o en su defecto habérselo saltado.
Poscondiciones		Los jugadores visualizan el tablero de juego en la pantalla.
Subcaso(s)		Colocar tropas y Turnos de juego.

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	El usuario selecciona la opción de “Jugar” ingresando al juego, pasando por todos los subcasos que representa dicho caso.	1	El sistema imprime en la pantalla del usuario un tablero de juego de 20 casillas de largo, con 90 casillas de ancho, dividido por un abismo.	Ninguna

Flujo de caso de uso

Excepciones: Este caso de uso no cuenta con ninguna excepción

Caso 6 (Subcaso del caso 5): Colocar tropas

Descripción del caso de uso

Actor(es)	Jugador 1 – 2.
Descripción	Utilizando las teclas “w”, “a”, “s” y “d”. El jugador puede mover el cursor para indicar en que posición del tablero pondrá a uno de sus soldados dándole a la tecla “Enter”. No se cuenta con un tiempo límite para colocar a las tropas por turno.
Precondiciones	<ul style="list-style-type: none"> • El usuario debe conocer cómo funciona el sistema. • El usuario debe conocer cómo poner una tropa en una casilla válida del tablero. • Se debe dar a conocer al usuario cuál de los dos jugadores está colocando una tropa.
Poscondiciones	<ul style="list-style-type: none"> • Los jugadores colocan uno de sus soldados en su parte del tablero de juego. • Al terminar de colocarse los soldados de un jugador, se habilita al otro jugador a colocar.
Subcaso(s)	Colocar tropas y Turnos de juego.

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	El primer jugador mueve usando “w”, “a”, “s”, “d” la posición donde desea colocar a su soldado.	1	<p>El cursor de la posición se actualiza cada vez que el jugador presiona una tecla.</p> <ul style="list-style-type: none"> • “W” mueve hacia arriba el cursor una casilla. • “A” mueve hacia la izquierda el cursor una casilla. • “S” mueve hacia abajo el cursor una casilla. • “D” mueve hacia la derecha el cursor una casilla. 	E1
2	EL jugador en turno presiona la tecla “Enter”.		En la pantalla, en la posición seleccionada, se dibuja un símbolo que representa al soldado que fue colocado. El sistema imprime en pantalla (bajo el tablero) en todo momento cuántos soldados quedan por colocar al jugador en turno.	E1, E2
3	El jugador en turno termina de colocar a todas sus tropas.		El sistema imprime que quedan pendientes 0 soldados, volviendo el flujo al paso uno de este caso para el siguiente jugador.	E1, E3

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	El usuario oprime la tecla "Esc".	Se ejecuta el caso 4.
E2	La casilla seleccionada no es válida.	El sistema no cola a ninguna tropa, tampoco reduce el contador de los soldados pendientes impreso en la pantalla, en su lugar mueve el cursor del usuario a una casilla válida.
E1	El contador de soldados pendientes para el primer jugador en turno no es 0.	El paso 1 y 2 de este caso se repiten hasta que se puede ejecutar correctamente el paso 3.

Caso 7 (Subcaso del caso 5): Turnos de juego

Descripción del caso de uso

Actor(es)		Jugador 1 – 2.
Descripción		El software detecta cuando uno de los usuarios ha terminado de actuar en su turno, entendiendo como acciones el disparo de cañón correspondiente a este, para cambiar de turno y habilitar las acciones del jugador contrario, esto de forma repetitiva hasta que se acabe la partida.
Precondiciones		<ul style="list-style-type: none"> • Deben haberse ejecutado la partida. • Deben haberse colocado los soldados tanto del jugador uno como del jugador dos.
Poscondiciones		El jugador en turno puede elegir la dirección y la potencia con la que disparará el proyectil en su turno.
Subcaso(s)		Disparo de proyectil, detección de proyectiles.

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	El usuario inicia la partida.	1	Se muestra en la pantalla del usuario de cuál de los dos jugadores es el turno, además se habilita el disparo del cañón.	
2	El jugador en turno ajusta la dirección de disparo de su cañón.	2	Se ejecuta el caso 8 y caso 9.	E1, E2

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	El usuario oprime la tecla “Esc”.	Se ejecuta el caso 4
E2	Se detecta que el jugador al cual le corresponde el siguiente turno ya no cuenta	Se termina la ejecución de la partida y se ejecuta el Caso 11.

	con tropas de su lado del tablero.	
--	---------------------------------------	--

Caso 8 (Subcaso del caso 7): Disparo de proyectil

Descripción del caso de uso

Actor(es)		Jugador 1 – 2.
Descripción	Durante la ejecución de la partida y en cada uno de los turnos, el jugador en turno podrá elegir la dirección a la cuál desea disparar su proyectil, así como la potencia.	
Precondiciones	<ul style="list-style-type: none"> • Deben haberse ejecutado la partida. • Deben haberse colocado los soldados tanto del jugador uno como del jugador dos. • Debe estarse ejecutando el Caso 7 “Turnos de juego”. 	
Poscondiciones	El jugador en turno dispara su proyectil, este se desplaza sobre la pantalla.	
Subcaso(s)	Ninguno.	

Actor			Sistema	
Paso	Acción	Paso	Acción	Excepción
1	El jugador en turno selecciona la dirección a la cuál desea disparar su proyectil con el cañón, utilizando las flechas de arriba y abajo.	1	El sistema detecta las entradas de las flechas y muestra en pantalla el inicio de la trayectoria que seguirá el disparo en caso de realizarse.	E1
2	Se oprime la tecla “Enter” y se elige la potencia del disparo.	2	El sistema imprime debajo del tablero de juego un menú para seleccionar la potencia de disparo. La potencia estará representada como números del 1 al 11. Siendo el 1 la potencia más baja y el 11 la más alta.	E1
3	Se oprimen las teclas “A” o “D”		El sistema detecta la entrada de las teclas, moviéndose entre las opciones de la siguiente manera:	E1

			<ul style="list-style-type: none"> • “A” Disminuye la potencia del tiro. • “D” Aumenta la potencia de tiro 	
4	El jugador em turno oprime la tecla “Enter”.		El software reconoce la potencia seleccionada por el usuario. Se ejecuta la animación del disparo del proyectil por el cañón, hasta que este haga impacto en las casillas del lado contrario del tablero de juego, es decir, el lado del jugador contrincante. Aquí se ejecuta el Caso 9.	

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	El usuario oprime la tecla “Esc”.	Se ejecuta el caso 4

Caso 9 (Subcaso del caso 7): Detección de proyectiles

Descripción del caso de uso

Actor(es)	Jugador 1 – 2.
Descripción	El sistema detecta la colisión de un proyectil lanzado en cualquiera de los turnos, ya sea con casillas vacías o en caso de tener un soldado enemigo, con casillas con una entidad.
Precondiciones	<ul style="list-style-type: none"> • Deben haberse ejecutado la partida. • Deben haberse colocado los soldados tanto del jugador uno como del jugador dos. • Debe haberse realizado un disparo de cañón por cualquiera de los dos jugadores.
Poscondiciones	El sistema detecta si ocurrió una colisión del proyectil con una entidad, eliminando el “Sprite” del o de los soldados eliminados.
Subcaso(s)	Determinar ganador.

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	El usuario dispara un proyectil con el cañón, como cualquiera de los dos jugadores.	1	Al colisionar el cañón con una parte del tablero, dibujando en pantalla círculos en aquellas casillas afectadas por el proyectil lanzado, siendo la casilla de impacto y las que la rodean; además el sistema analiza si hay una tropa de un jugador en dichas casillas (donde hubo una colisión directa y donde hubo colisiones indirectas), si es así, cambia su “Sprite” con una letra X.	E1
2	El usuario recibe una retroalimentación de lo sucedido en la ejecución.		El sistema reduce el número de soldados del jugador que se vio afectado por el tiro de su rival en base al número de “sprites” que hayan sido cambiados por la letra “X”. Esta información se imprime en pantalla para la lectura del usuario. Al terminarse se ejecuta el caso 7.	E1

Flujo de caso de uso

Excepciones

Identificador	Nombre	Acción
E1	El usuario oprime la tecla “Esc”.	Se ejecuta el caso 4

Caso 10 (Subcaso del caso 7): Determinar ganador de la partida

Descripción del caso de uso

Actor(es)	Jugador 1 – 2.
Descripción	El software detecta cuando un jugador se quedó sin ningún soldado de su lado del tablero, de esta forma el jugador contrario a este será el ganador.
Precondiciones	<ul style="list-style-type: none"> • Deben haberse ejecutado la partida. • Deben haberse colocado los soldados tanto del jugador uno como del jugador dos. • Debe haberse realizado un disparo de cañón por cualquiera de los dos jugadores. • Debe haberse detectado la colisión o no colisión del proyectil lanzado.
Poscondiciones	El sistema termina la partida e imprime quien fue el ganador.
Subcaso(s)	Ninguno.

Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	Uno de los jugadores entra nuevamente en su turno a partir del caso 7.	1	El sistema analiza la cantidad de soldados que le quedan a dicho jugador, si la cantidad de soldados del lado de su tablero es igual a 0, entonces el sistema termina vuelve al Suceso E2 del caso 7, terminando la ejecución de la partida.	

Flujo de caso de uso

Excepciones: Este caso de uso no cuenta con ninguna excepción

Caso 11: FeedBack

Descripción del caso de uso

Actor(es)		Jugador 1 – 2.
Descripción	Después de terminar la partida el sistema imprime al usuario las estadísticas de juego del jugador uno y el jugador dos.	
Precondiciones	<ul style="list-style-type: none"> • Deben haberse ejecutado la partida. • Deben haberse colocado los soldados tanto del jugador uno como del jugador dos. • Debe haberse realizado un disparo de cañón por cualquiera de los dos jugadores. • Debe haberse detectado la colisión o no colisión del proyectil lanzado. • Debe haberse determinado el ganador de la partida. 	
Poscondiciones	Se imprimen en pantalla las estadísticas de los dos jugadores.	
Subcaso(s)	Ninguno.	




Actor		Sistema		
Paso	Acción	Paso	Acción	Excepción
1	Los jugadores terminan su partida	1	Se imprime quien de los dos jugadores fue el ganador. Además, se imprime en pantalla las estadísticas del jugador uno, así como las del jugador número dos.	
2	Se presiona la tecla "Enter".		El software devuelve el flujo al Caso 1, volviendo al menú principal.	



Flujo de caso de uso


Excepciones: Este caso de uso no cuenta con ninguna excepción.



11.0 Matriz de Requerimientos



Requerimientos funcionales y no funcionales:	Objetivos específicos:					
	1	2	3	4	5	6
RF-001. El sistema deberá contar con un menú principal de opciones que se le muestre en pantalla al usuario al iniciar la ejecución del software. Dicho menú tendrá las siguientes opciones: <ol style="list-style-type: none"> 1. Jugar 2. Configurar audio 3. Créditos 4. Salir 				✓		
RF-002. Menú de audio Eligiendo el botón con la opción de “Configurar audio” en el menú principal, el sistema permite a los usuarios ingresar al “menú de audio”, un apartado en donde se verá la barra de nivel de volumen, este servirá para ajustar las configuraciones de sonido del juego.					✓	✓
RF-003. Configurar música En el “menú de audio” existe una barra lateral que permite subir, bajar o silenciar por completo el volumen de la música del sistema, este está barra lateral se tendrá un círculo para elegir el nivel de volumen						✓





que se desee; si dicho círculo se desplaza hacia la izquierda, se bajará el nivel del volumen, de lo contrario, se aumentará el nivel del volumen.						
<p>RF-004</p> <p>Configurar efectos de sonido</p> <p>En el “menú de audio” existe una barra lateral que permite subir, bajar o silenciar por completo el volumen de los efectos de sonido del sistema, este está barra lateral se tendrá un círculo para elegir el nivel de volumen que se desee; si dicho círculo se desplaza hacia la izquierda, se bajará el nivel del volumen, de lo contrario, se aumentará el nivel del volumen.</p>						
<p>RF-005.</p> <p>Ingresar al juego</p> <p>Los usuarios podrán ingresar a la ejecución de una partida del juego usando la opción de “jugar” en el menú principal, también podrían ingresar a la ejecución de la partida presionando la tecla “Enter” de su ordenador de mesa o portátil en el mismo menú. El sistema redirige a los usuarios a la pantalla de juego iniciando una partida directa.</p>						





<p>RF-006.</p> <p>Salir del sistema</p> <p>El sistema permite que el usuario pueda salir de este último usando el botón de “salir” ubicado en el menú principal, al presionarlo automáticamente se cerrará el juego, dejando de ejecutarlo.</p>						
<p>RF-007.</p> <p>Tutorial</p> <p>Al iniciar la ejecución de la partida mediante el botón “jugar”, independientemente de si se eligieron aspectos o no para los personajes, el software despliega en pantalla del usuario un tutorial que le explica cómo funcionan las principales mecánicas del juego:</p> <ul style="list-style-type: none"> • Al seleccionar las tropas <ul style="list-style-type: none"> ○ Flecha arriba / W: Desplazar la tropa un espacio arriba ○ Flecha abajo / S: Desplazar la tropa un espacio abajo ○ Flecha derecha / D: Desplazar la tropa un espacio a la derecha ○ Flecha izquierda / A: Desplazar 						


<p>la tropa un espacio a la izquierda</p> <ul style="list-style-type: none"> ○ Enter: Colorar la tropa en el espacio situado ● Al direccionar el cañón <ul style="list-style-type: none"> ○ Flecha arriba / W: Direccionar el cañón hacia arriba ○ Flecha abajo / S: Direccionar el cañón hacia abajo ○ Enter: Seleccionar posición del cañón ● Al potenciar el cañón <ul style="list-style-type: none"> ○ Flecha derecha / D: Incrementar la potencia ○ Flecha izquierda / A: Decrementar la potencia ○ Enter: Seleccionar la potencia del cañón 						
<p>RF-008.</p> <p>Objetivo de destrucción</p> <p>Durante la ejecución del tutorial se deberá establecer al o los usuarios que como objetivo principal del juego se debe buscar la destrucción de objetos, es decir, la eliminación de objetivos específicos (las</p>						




tropas enemigas) mediante los lanzamientos de proyectiles, mencionando que el objetivo busca derribar todas las tropas enemigas para ganar.						
<p>RF-009.</p> <p>Menú de selección de aspecto.</p> <p>Después de seleccionar la opción “jugar” en el menú principal, pero antes de iniciar una partida, el sistema mostrará en pantalla una ventana que pregunta si se desea ingresar al menú de selección de aspecto. Si el usuario selecciona la opción “sí” se mostrará en pantalla dicho menú para que los jugadores pueden elegir el aspecto físico que tendrán sus tropas en el tablero de juego. En dicho menú estarán las siguientes opciones:</p> <p>Cambiar aspectos del jugador 1.</p> <p>Cambiar aspectos del jugador 2.</p> <p>Aceptar.</p> <p>Al elegir la opción de “Aceptar” el menú se cerrará y se terminará la selección de aspecto para sus personajes.</p>						
<p>RF-010.</p> <p>Detectar la selección de aspectos.</p> <p>Cuando se selecciona un aspecto por parte del usuario el</p>						





sistema detecta cuál de los jugadores ha seleccionado el aspecto de sus tropas, deshabilitando este aspecto para el jugador contrario al que selecciono.						
<p>RF-011.</p> <p>Cambiar el aspecto para el primer o segundo jugador</p> <p>Se selecciona la opción “Cambiar aspectos del jugador 1” o en su defecto “Cambiar aspectos del jugador 2”. El software despliega en pantalla para los usuarios una galería con todos los aspectos disponibles, el usuario puede desplazar de izquierda a derecha en esta galería dando “click” con el cursor del “ratón” a unos botones ubicados a los lados de la pantalla (estos tendrán flechas indicando la dirección a la que mueven la galería). Cuando tenga seleccionado el aspecto que desee y al darle con el cursor a la opción de “aceptar” centrada en la parte inferior, se habrá completado el cambio de aspecto y se regresará al “menú de selección de aspecto”.</p>						
<p>RF-012.</p> <p>Interacción multijugador local.</p> <p>Al iniciar la ejecución de una partida del juego, es decir, ya que se haya seleccionado la opción de “jugar” y se haya</p>						




mostrado el tutorial, el software permite a los jugadores interactuar entre ellos en un entorno multijugador local de 1v1, asignándole turnos de acción al jugador uno y al jugador dos, de forma que el usuario que maneja el sistema se convertirá en un turno en el primer jugador, y en el siguiente será el segundo jugador, evitando que ambos bandos interactúen en el mismo momento.						
<p>RF-013.</p> <p>Tablero de juego</p> <p>El software dentro de la escena designada para la ejecución de las partidas (donde los usuarios pueden jugar) se cuenta con un tablero de juego de 20 casillas en forma de filas con 90 casillas en forma de columnas, este tablero se le mostrará en la pantalla al usuario. Deberá dividirse a la mitad por una franja de casillas donde ninguno de los dos jugadores tendrá permitido el despliegue de tropas, esta franja inhábil de casillas estará designada para ser el abismo que divide el tablero, el cual da nombre al juego.</p>						
<p>RF-014.</p> <p>Colocar tropas</p>						






Los usuarios durante la ejecución de la partida pueden elegir donde colocar a sus 12 soldados en las casillas validas del tablero de juego. Esta acción se realizará al iniciar la partida, empezando con el jugador uno, en el que tendrá que colocar todas sus tropas a su elección para que el siguiente jugador realice la misma acción.						
<p>RF-015.</p> <p>Potencia de proyectiles</p> <p>Se implementa una mecánica, durante la ejecución de una partida de juego, que permita cargar la potencia de un proyectil al elegir dicha potencia en un intervalo del 1 al 11, siendo el 1 la potencia más baja y el 11 la potencia más alta. Se podrá elegir utilizando las flechas izquierda y derecha, o los botones “A” y “D” para cambiar el valor, se establecerá la potencia elegida por el usuario dándole “click” al botón “Enter”.</p>						
<p>RF-016.</p> <p>Trayectoria de proyectiles</p> <p>Se calcula de forma interna la trayectoria que llevará el proyectil que será lanzado, mostrando en la pantalla para el jugador en turno el inicio de esta trayectoria previamente calculada, evitando que se le</p>						





<p>muestre el total de la trayectoria al usuario, la trayectoria que se verá será del lado del jugador en turno, mostrando con flechas hacia donde se dirigiría el proyectil, se puede cambiar la trayectoria hacia arriba o abajo utilizando los controles “W” / Flecha arriba o “S” / Flecha abajo respectivamente, se establecerá la trayectoria con el botón “Enter”.</p>						
<p>RF-017.</p> <p>Turnos de los jugadores</p> <p>Según los jugadores colocan a cada soldado, el software deberá mostrar en pantalla cuántos le faltan por colocar, así como detectar cuándo termine de colocar a todas sus tropas. Al terminar el tiempo designado, el sistema deberá cambiar el turno y actuar al segundo jugador, quien podrá colocar sus tropas igual que el primero. El software deberá ser capaz de reconocer cuando ambos jugadores hayan terminado de colocar sus tropas, así iniciando la ejecución normal del cambio de turnos, cambiando de turno del jugador uno al dos y viceversa cada vez que ellos realicen una acción, es decir, disparen un proyectil con su cañón, pero si cualquiera de los dos jugadores realiza un tiro exitoso en su cañón se le permitirá realizar otro tiro, así hasta que falle, al fallar</p>						

automáticamente se cambiará de turno al jugador rival.						
<p>RF-018.</p> <p>Daño de área</p> <p>Los proyectiles lanzados por el cañón de cualquiera de los usuarios no solo pueden eliminar una tropa enemiga por contacto directo, si no que deberán contar con daño de área para favorecer la inmersión en el juego. El daño de área podrá eliminar tropas enemigas que estén alrededor de donde impacto el proyectil que haya sido lanzado, en caso de no haber tropas enemigas alrededor, solo se eliminará a la que haya impactado directamente, o en su defecto, no se eliminará a ningún soldado enemigo en caso de fallar el disparo.</p>						
<p>RF-019.</p> <p>Detección de proyectiles</p> <p>El software detecta cuando un proyectil impacta directa o indirectamente (por daño de área) sobre una tropa enemiga durante la ejecución del turno de uno de los jugadores, eliminando el Sprite de dicha tropa de la pantalla, y reduciendo el número de tropas del jugador que perdió a su guerrero en uno. Al detectarse un impacto de un proyectil</p>						

<p>contra una tropa enemiga, el sistema añade permite realizar otro tiro en el turno. Si no sucede impacto, el tiempo del turno se acaba automáticamente y se pasa al siguiente turno. Al eliminar la tropa enemiga, automáticamente aparecerá una “X” marcada en el espacio donde se sitúa la tropa eliminada.</p>						
<p>RF-020.</p> <p>Determinar ganador</p> <p>Después del lanzamiento de cualquiera de los jugadores y de que el sistema haya detectado la colisión o no colisión del proyectil, también analiza y determina cuando uno de los dos jugadores se queda sin tropas en el tablero, en caso de que se cumpla esta condición entonces la partida termina en ese momento. Al terminarse la partida en pantalla se despliega cuál de los dos usuarios fue el ganador y se devuelve a la pantalla de menú principal.</p>						
<p>RF-021.</p> <p>FeedBack</p> <p>Al terminar la partida, se retroalimentará a los dos jugadores sobre el número de lanzamientos, derribos y tropas restantes que tuvo cada uno. Esta retroalimentación se imprimirá en pantalla al usuario en forma de ventana y se deberá seleccionar la opción de</p>						

“Aceptar” centrada en la parte inferior para terminar de forma oficial la ejecución de una partida del juego.						
RF-022. Pausar la partida El software cuenta con un botón de pausa durante la escena donde se desarrolla la partida entre los jugadores, deberá ubicarse en la esquina superior izquierda de la pantalla y permite desplegar una ventana opciones. Entre las opciones se encuentra “salir al menú principal” o reanudar la partida. Al seleccionar “Salir al menú principal” el sistema devuelve al usuario al menú principal.						
RF-023. Solución de problemas Se cuenta con una opción en el menú principal para contactar a los administradores para la solución de problemas bien para ingresar como un administrador. Si no es administrador, se deberá seleccionar la opción de ayuda y luego se desplegará en pantalla, para los usuarios, la información de contacto. Para esta opción será necesario que exista una conexión a internet.						
RF-024. Ingresar como administrador						

Los administradores pueden ingresar al sistema para favorecer a la resolución de problemas de los usuarios. Para ello necesitarán contar con conexión a internet, ingresar a la opción administradores del menú principal, posteriormente seleccionar “soy administrador” e ingresar su usuario y contraseña.						
RNF-001. El sistema debe estar optimizado para poder utilizarse en computadores que cuenten con un sistema operativo Windows 8 en adelante.						
RNF-002. El juego debe ser capaz de ejecutarse de manera fluida y sin retrasos en el dispositivo, manteniendo una tasa de “frames” por segundo (FPS) adecuada.						
RNF-003. Se debe optimizar para que funcione de forma fluida en computadoras con mínimo 4 GB de RAM en adelante.						
RNF-004. El software debe contar con un diseño de interfaces intuitivas y minimalista que permita a los usuarios navegar fácilmente entre sus opciones y que además ayude a tener un rápido aprendizaje de cómo utilizarlo.						

<p>RNF-005.</p> <p>Los controles del juego responden de manera precisa y segura a las acciones del jugador, reaccionando a las acciones que éste decida</p>						
<p>RNF-006.</p> <p>Debe estar optimizado para su ejecución de forma local en ordenadores de sobre mesa y portátiles (laptops), es decir, no se debe requerir que exista una conexión a internet para poder utilizar el sistema.</p>						
<p>RNF-007.</p> <p>El sistema implementa gráficos y animaciones retro en 2D para que los jugadores tengan mayor inmersión, con el objetivo de reforzar la temática medieval del juego.</p>						
<p>RNF-008.</p> <p>Cuenta con efectos de sonidos y música ambiental (“soundtrack”). La música ambiental debe cambiar por cada escenario, es decir, será diferente para los menús con respecto a la ejecución de la partida.</p>						

12.0 Proceso de Desarrollo

12.1 Herramientas utilizadas

12.1.1 Dev C++

Se utiliza este entorno de desarrollo integrado para programar la lógica del producto de software en lenguaje C, esto porque es el programa estándar con el que se trabajó durante el curso. Adicionalmente se usarán la biblioteca externa freeglut.h para habilitar funciones de OpenGL y poder añadir ventanas y gráficos.

12.1.3 Figma

Se utiliza Figma para la elaboración de las interfaces de usuario de baja y alta fidelidad para los módulos del producto de software.

12.1.4 GitHub – GitKraken

Se utilizan estos programas para llevar un control de versiones, tanto del código como de la documentación asociada a este. Se realiza en un repositorio privado y es parte de la organización interna del equipo.

12.1.5 WhatsApp

Herramienta de comunicación en el que los integrantes del equipo mantienen conversaciones e ideas para el seguimiento del proyecto colaborativo, además de compartir las demás herramientas y links utilizados para dicho proyecto.

12.1.6 Discord

Herramienta de comunicación en el que los integrantes del equipo se reúnen de manera virtual para las videoconferencias.

12.1.7 Canva

Página en el que se utiliza para la elaboración de “canvas” o presentaciones visuales para alimentar la información proporcionada de una manera más ilustrativa.

12.2 Organización

La organización se realizará de manera informal primero para garantizar mayor fluidez y rapidez, y después se organizarán las tareas a realizar más formalmente.

12.2.1 WhatsApp – Discord

Los primeros canales de comunicación del equipo y los más informales, son las aplicaciones en donde se organiza colectivamente qué parte realizará cada quien en las etapas del proyecto, con respecto a la documentación, codificación, presentación, entre otros.

- Documentación
 - Alonzo Palacios Rodrigo
 - Martínez Martínez José Pablo
- Codificación
 - Cuevas García Braulio Samuel
 - Moo Pan Jareth Jaziel

12.2.2 To do List

Almacenado en el repositorio privado en de este proyecto en Github, se encuentra la “To do list” en la en el tablero Kanban. Nos sirve para registrar las tareas del desarrollo pendientes, las urgentes, identificar las que se encuentran en proceso, y visualizar las que ya fueron completadas con éxito.

Listado de deberes a realizar en Kanban	
Alonzo Palacios Rodrigo	<ul style="list-style-type: none"> • Mapeo y matriz de requerimientos • Casos de uso • Modularidad • Presentación
Cuevas García Braulio Samuel	<ul style="list-style-type: none"> • Desarrollo de interfaces • Investigación de código • Estándar de codificación
Martínez Martínez José Pablo	<ul style="list-style-type: none"> • Mapeo y matriz de requerimientos • Casos de uso • Diagrama de casos de uso • Estándar de codificación
Moo Pan Jareth Jaziel	<ul style="list-style-type: none"> • Trabajo de codificación • Implementación del código

12.3 Monitoreo

12.3.1 *GitHub – GitKraken*

Cómo se mencionó anteriormente, se utiliza Github y Gitkraken para llevar un control de versiones, tanto del código como de la documentación asociada a este. Además de las funciones anteriormente mencionadas, también permiten guardar los procesos y participaciones individuales en forma del número de “pull request” de cada integrante del equipo, de esta forma sirve como control de versiones.

12.4 Bitácoras

12.4.1 *Microsoft To Do*

Se especificará una fecha y hora máxima para realizar las actividades correspondientes de la elaboración de los proyectos asignados a cada persona para evitar contratiempos. Esta información se podrá encontrar de manera específica para cada tarea pendiente en la “to do list” ubicada en el repositorio del proyecto.

<input type="radio"/>	Estándar de codificación Tareas • 📅 dom., 28 de abr.	☆
<input type="radio"/>	Modularidad Tareas • 📅 sáb., 27 de abr.	☆
▼ Completado 8		
<input checked="" type="radio"/>	Trabajo de codificación Tareas • 📅 dom., 28 de abr.	☆
<input checked="" type="radio"/>	Corregir errores de codificación Tareas • 📅 lun., 29 de abr.	☆
<input checked="" type="radio"/>	Implementación del código Tareas • 📅 lun., 29 de abr.	☆
<input checked="" type="radio"/>	Investigación del código Tareas • 📅 vie., 26 de abr.	☆
<input checked="" type="radio"/>	Desarrollo de interfaces Tareas • 📅 vie., 26 de abr.	☆
<input checked="" type="radio"/>	Diagrama de casos de uso Tareas • 📅 dom., 28 de abr.	☆
<input checked="" type="radio"/>	Casos de uso Tareas • 📅 sáb., 27 de abr.	☆
<input checked="" type="radio"/>	Mapeo de requerimientos Tareas • 📅 jue., 25 de abr.	☆

12.5 Medición del trabajo grupal

El trabajo grupal y el desempeño del equipo se determinarán mediante reuniones diarias dirigidas por un SCRUM máster o en su defecto el líder de equipo, en estas reuniones se compartirán los avances de cada integrante, se expondrán dudas y se analizará cómo se progresó según la bitácora, si hay un problema se busca identificarlo y solucionarlo. Además, se procurará tomar en cuenta opiniones o cualquier aporte dado para la implementación y mejora del proyecto por parte de un integrante.

12.6 Medición del trabajo individual

Haciendo uso de la bitácora y de la actividad registrada por persona en el repositorio del proyecto, se medirá el trabajo individual. A lo mencionado, se le añadirá el cumplimiento de las actividades no relacionadas con la codificación asignadas a cada integrante del equipo, analizando si se cumplieron en tiempo y forma con respecto a los tiempos establecidos en la bitácora.

Listado de deberes	Persona(s) asignada(s)	Porcentaje	Individual
Mapeo y matriz de requerimientos	<ul style="list-style-type: none"> Alonzo Palacios Rodrigo Martínez Martínez José Pablo 	15%	<ul style="list-style-type: none"> 7.5% 7.5%
Casos de uso	<ul style="list-style-type: none"> Alonzo Palacios Rodrigo Martínez Martínez José Pablo 	10%	<ul style="list-style-type: none"> 5% 5%
Modularidad	<ul style="list-style-type: none"> Alonzo Palacios Rodrigo 	5%	<ul style="list-style-type: none"> 5%
Trabajo de codificación	<ul style="list-style-type: none"> Moo Pan Jareth Jaziel 	25%	<ul style="list-style-type: none"> 25%
Desarrollo de interfaces	<ul style="list-style-type: none"> Cuevas García Braulio Samuel 	5%	<ul style="list-style-type: none"> 5%
Investigación del código	<ul style="list-style-type: none"> Cuevas García Braulio Samuel 	5%	<ul style="list-style-type: none"> 5%
Diagrama de casos de uso	<ul style="list-style-type: none"> Martínez Martínez José Pablo 	10%	<ul style="list-style-type: none"> 10%
Estándar de codificación	<ul style="list-style-type: none"> Martínez Martínez José Pablo Cuevas García Braulio Samuel 	10%	<ul style="list-style-type: none"> 5% 5%
Implementación del código	<ul style="list-style-type: none"> Moo pan Jareth Jaziel 	5%	<ul style="list-style-type: none"> 5%
Corrección de errores en codificación	<ul style="list-style-type: none"> Cuevas García Braulio Samuel 	5%	<ul style="list-style-type: none"> 5%
Presentación	<ul style="list-style-type: none"> Alonzo Palacios Rodrigo 	5%	<ul style="list-style-type: none"> 5%
Porcentaje total: 100%			

Porcentaje individual:

- **Alonzo Palacios Rodrigo:** 22.5%
- **Cuevas García Braulio Samuel:** 20%
- **Martínez Martínez José Pablo:** 27.5%
- **Moo Pan Jareth Jaziel:** 30%

13.0 Primer avance de código

En el primer reporte de avance se aprecian las funciones principales que se creía necesitaría para el funcionamiento del sistema, pero, según evolucionaba el desarrollo del proyecto, se tuvo que cambiar o modificar casi todas las funciones presentadas a continuación. En el siguiente reporte.

Estas eran las funciones que permitían visualizar los menús, tanto el principal como el secundario (dentro de una partida).

```
201
202 // Función para mostrar el menú principal
203 void show_main_menu() {
204     printf("MENU PRINCIPAL\n");
205     printf("1. Iniciar juego\n");
206     printf("2. Cerrar programa\n");
207     printf("Seleccione una opción: ");
208 }
209
210 // Función para mostrar el menú secundario
211 void show_submenu() {
212     printf("\nMENU SECUNDARIO\n");
213     printf("2. Reanudar partida\n");
214     printf("3. Regresar al menú principal\n");
215     printf("Seleccione una opción: ");
216 }
217
```

Función para mostrar una representación visual de una tropa en el tablero.

```
// Función para mostrar una representación visual de una tropa en el tablero
void visualize_troop(char board[BOARD_HEIGHT][BOARD_WIDTH], int row, int col, Troop *troop) {
    char symbol = '#'; // Representa una tropa
    if (troop->vertical) {
        for (int i = 0; i < troop->size; i++) {
            board[row + i][col] = symbol;
        }
    } else {
        for (int i = 0; i < troop->size; i++) {
            board[row][col + i] = symbol;
        }
    }
    print_board(board, "ABYSS'S BATTLE");
}
```

Función que leía las tropas ingresadas por el usuario para acomodarlas en una casilla válida en el tablero.

```
// Función para que el jugador coloque sus tropas
void place_troops_manually(char board[BOARD_HEIGHT][BOARD_WIDTH], Troop *troops) {
    for (int i = 0; i < NUM_TROOPS; i++) {
        printf("Coloca la tropa de tamaño %d (formato: fila columna orientación (V/H)): ", troops[i].size);
        int row, col;
        char orientation;
        scanf("%d %d %c", &row, &col, &orientation);
        row--; // Ajustamos al índice base 0
        col--; // Ajustamos al índice base 0
        orientation = toupper(orientation);
        if (!is_valid_coordinate(row, col) || (orientation != 'V' && orientation != 'H')) {
            printf("Coordenadas inválidas. Por favor, inténtalo de nuevo.\n");
            i--;
            continue;
        }
        troops[i].vertical = (orientation == 'V');
        if (!can_place_troop(board, row, col, &troops[i])) {
            printf("La tropa no cabe en esta posición o se superpone con otra. Por favor, inténtalo de nuevo.\n");
            i--;
            continue;
        }
        visualize_troop(board, row, col, &troops[i]);
        place_troop(board, row, col, &troops[i]);
    }
}
```

Función que colocaba las tropas en el tablero.

```
// Función para colocar una tropa en el tablero
void place_troop(char board[BOARD_HEIGHT][BOARD_WIDTH], int row, int col, Troop *troop) {
    if (troop->vertical) {
        for (int i = 0; i < troop->size; i++) {
            board[row + i][col] = '#'; // Representa una tropa
        }
    } else {
        for (int i = 0; i < troop->size; i++) {
            board[row][col + i] = '#'; // Representa una tropa
        }
    }
}
```

Función para que el jugador realice un ataque (esta estaba basada en la lógica implementada en juegos similares).

```
// Función para que el jugador realice un ataque
void player_attack(char board[BOARD_HEIGHT][BOARD_WIDTH], char opponent_board[BOARD_HEIGHT][BOARD_WIDTH], Troop *troops, bool *extra_shot) {
    int row, col;
    printf("Introduce las coordenadas de tu ataque (fila columna): ");
    scanf("%d %d", &row, &col);
    row--; // Ajustamos al índice base 0
    col--; // Ajustamos al índice base 0

    if (!is_valid_coordinate(row, col)) {
        printf("Coordenadas inválidas. Por favor, inténtalo de nuevo.\n");
        player_attack(board, opponent_board, troops, extra_shot); // Pedir coordenadas nuevamente
        return;
    }

    if (opponent_board[row][col] == '0') {
        printf("¡Impacto!\n Puede volver a atacar \n");
        board[row][col] = 'X';
        opponent_board[row][col] = 'X';
        // Marcar la tropa como golpeada
        for (int i = 0; i < NUM_TROOPS; i++) {
            if (troops[i].vertical) {
                if (col >= col && col < col + troops[i].size && row >= row && row < row + troops[i].size) {
                    troops[i].hit[col - row] = true;
                    // Comprobar si la tropa ha sido eliminada
                    bool eliminated = true;
                    for (int j = 0; j < troops[i].size; j++) {
                        if (!troops[i].hit[j]) {
                            eliminated = false;
                            break;
                        }
                    }
                    if (eliminated) {
                        printf("Has eliminado una tropa enemiga!\n");
                    }
                }
            }
        }
    }
}
```

```
        }
    } else {
        if (row >= row && row < row + troops[i].size && col >= col && col < col + troops[i].size) {
            troops[i].hit[row - col] = true;
            // Comprobar si la tropa ha sido eliminada
            bool eliminated = true;
            for (int j = 0; j < troops[i].size; j++) {
                if (!troops[i].hit[j]) {
                    eliminated = false;
                    break;
                }
            }
            if (eliminated) {
                printf("Has eliminado una tropa enemiga!\n");
            }
            break;
        }
    }
}

*extra_shot = true; // Dar un disparo adicional
} else if (opponent_board[row][col] == 'X' || opponent_board[row][col] == '0') {
    printf("Ya has atacado esta posición. Por favor, elige otra.\n");
    player_attack(board, opponent_board, troops, extra_shot); // Pedir coordenadas nuevamente
} else {
    printf("Fallo...\n");
    board[row][col] = '0';
    opponent_board[row][col] = '0';
}
}
```

Función que se planeaba usar para verificar si uno de los dos jugadores había ganado.

```
// Función para verificar si un jugador ha ganado
bool check_win(char board[BOARD_HEIGHT][BOARD_WIDTH]) {
    for (int i = 0; i < BOARD_HEIGHT; i++) {
        for (int j = 0; j < BOARD_WIDTH; j++) {
            if (board[i][j] == '#') {
                return false;
            }
        }
    }
    return true;
}
```

Función Main principal.

```
int main() {
    setlocale(LC_ALL, "");
    char player1_board[BOARD_HEIGHT][BOARD_WIDTH];
    char player2_board[BOARD_HEIGHT][BOARD_WIDTH];
    char player1_shots[BOARD_HEIGHT][BOARD_WIDTH];
    char player2_shots[BOARD_HEIGHT][BOARD_WIDTH];

    // Inicializar tableros
    init_board(player1_board);
    init_board(player2_board);
    init_board(player1_shots);
    init_board(player2_shots);

    // Definir las tropas de cada jugador
    Troop player1_troops[NUM_TROOPS] = {{5, true}, {4, true}, {3, true}, {3, true}, {2, true}};
    Troop player2_troops[NUM_TROOPS] = {{5, true}, {4, true}, {3, true}, {3, true}, {2, true}};

    int main_menu_option;
    do {
        show_main_menu();
        scanf("%d", &main_menu_option);
        switch (main_menu_option) {
            case 1: {
                // Colocar tropas manualmente
                printf("Jugador 1, coloca tus tropas:\n");
                place_troops_manually(player1_board, player1_troops);
                printf("Jugador 2, coloca tus tropas:\n");
                place_troops_manually(player2_board, player2_troops);

                // Juego
                bool game_over = false;
                bool player1_turn = true;
                bool player1_extra_shot = false;
                bool player2_extra_shot = false;
                while (!game_over) {
                    print_board(player1_board, "Tablero del Jugador 1:");
                }
            }
        }
    } while (true);
}
```

Bucle principal del juego.

```
place_troops_manually(player2_board, player2_troops);

// Juego
bool game_over = false;
bool player1_turn = true;
bool player1_extra_shot = false;
bool player2_extra_shot = false;
while (!game_over) {
    print_board(player1_board, "Tablero del Jugador 1:");
    print_board(player2_board, "Tablero del Jugador 2:");

    if (player1_turn) {
        printf("\nTurno del Jugador 1:\n");
        player_attack(player1_shots, player2_board, player2_troops, &player1_extra_shot);
        game_over = check_win(player2_board);
        if (!player1_extra_shot) {
            player1_turn = false;
        }
        player1_extra_shot = false;
    } else {
        printf("\nTurno del Jugador 2:\n");
        player_attack(player2_shots, player1_board, player1_troops, &player2_extra_shot);
        game_over = check_win(player1_board);
        if (!player2_extra_shot) {
            player1_turn = true;
        }
        player2_extra_shot = false;
    }
}
```


Visualizar el Menú secundario durante la partida.

```
// Mostrar el menú secundario
if (!game_over) {
    show_submenu();
    int submenu_option;
    scanf("%d", &submenu_option);
    switch (submenu_option) {
        case 1:
            // Reiniciar partida
            init_board(player1_board);
            init_board(player2_board);
            init_board(player1_shots);
            init_board(player2_shots);
            place_troops_manually(player1_board, player1_troops);
            place_troops_manually(player2_board, player2_troops);
            game_over = false;
            player1_turn = true;
            break;
        case 2:
            // Reanudar partida (no hace nada)
            break;
        case 3:
            // Regresar al menú principal
            game_over = true;
            break;
        default:
            printf("Opción inválida. Por favor, seleccione nuevamente.\n");
            break;
    }
}
}
```

Cierre del programa.

```
break;
}
}

printf("Juego terminado! ");
if (player1_turn) {
    printf("El Jugador 1 ha ganado!\n");
} else {
    printf("El Jugador 2 ha ganado!\n");
}
break;
}

case 2:
    printf("Hasta luego!\n ---CREDITOS--- \n TEAM CFORCE \n Alonzo Palacios Rodrigo Alonzo \n Cuevas Garcia Braulio Samuel \n Martinez Martin");
    return 0;
default:
    printf("Opción inválida. Por favor, seleccione nuevamente.\n");
    break;
}
} while (true);

return 0;
```

14.0 Reporte final del código

Funcionalidad de los requerimientos implementados

RF-001: Menú principal.

Función principal (Main): Esta contiene la lógica para acceder a la impresión del menú principal, así como a las otras opciones disponibles:

1. Función para mostrar los créditos.
2. Función para iniciar una partida del juego (“iniciarjuego”).

RF-006: Salir del sistema.

3. Salir del programa. Se encuentra entre las opciones que se pueden elegir en el menú principal. Al elegirse se termina automáticamente la ejecución de todo el programa.

Observación:

Función para ajustar el audio. Esta función no se desarrolló ya que por falta de tiempo no se pudo implementar la ejecución de audio ni animaciones en el proyecto, por ello no se realizaron los requerimientos enfocados a dichos rubros, siendo este el motivo por el cuál no serán mencionados en este apartado y tampoco hay funciones enfocadas a ellos en el código del proyecto.

RF-005: Ingresar al juego.

La función principal de este requerimiento en el código del sistema lleva por nombre “iniciarjuego”. Desde esta se hace la ejecución de los subprocesos necesarios para la correcta ejecución de una partida.

RF-007: Tutorial.

En la misma función principal donde se imprime la retroalimentación a los usuarios de que han iniciado a jugar, se imprime en pantalla de forma breve pero clara la información que los jugadores necesitan para usar el software durante una partida. La implementación es diferente a la que se tenía planificada en el requerimiento, ya que se le explican las instrucciones para usar el software durante la ejecución de una partida al usuario de forma más breve y sin preguntar si las desea o no, esto sucedió por que se consideró más importante enfocarse en mejorar las mecánicas principales de juego.

Función “iniciarjuego”, desde la cual se hace la llamada al resto de las funciones que permiten el correcto funcionamiento del software en el momento en que se ejecuta una partida entre dos jugadores:

```
//LLAMADA A LAS OTRAS FUNCIONES DEL PROCESO

int iniciarjuego() //Funcion principal para inicializar el juego, iniciando en ella los turnos.
{
    setlocale(LC_ALL, "");

    char tablero[FILAS][COLUMNAS];
    char keyMain; int moment=1, c1=0, c2=0, potencia;
    player jugador[2];
    //Establece que jugador comienza:
    jugador[0].turno=1;
    jugador[1].turno=0;
    jugador[0].disparos=0;
    jugador[1].disparos=0;

    inicializarTablero(tablero);
    setCanon(jugador, tablero);
    setSoldier1(8jugador[0], tablero); setSoldier1(8jugador[1], tablero);

    do {
        imprimirTableroAux(tablero);
        switch(moment){
            case 1:
                //if turno jugador 1
                printf("\nJugador 1, coloca a tus soldados\n\n"); //Explicaciones para el usuario.
                printf("\tMueve el cursor con las flechas o con WASD\n");
                printf("\tPara seleccionar la posicion, presiona enter\n");
                inputSet(8jugador[0], tablero, 8keyMain, 8c1);
                //if c1 = 4 cambiar turnos y aumentar momento
                if (c1==MAX_SOLDIER){
                    switchTurno(8jugador[0], 8jugador[1]);
                    moment++;
                    c1=0;
                }
                break;
            case 2:
                printf("\nJugador 2, coloca a tus soldados\n\n");
                printf("\tMueve el cursor con las flechas o con WASD\n");
                printf("\tPara seleccionar la posicion, presiona enter\n");
                inputSet(8jugador[1], tablero, 8keyMain, 8c1);
                if (c1==MAX_SOLDIER){
                    switchTurno(8jugador[0], 8jugador[1]);
                    moment++;
                    system("cls");
                }
                break;
        }
    } while (1);
}
```

```

        setAim(&jugador[0]);
        setAim(&jugador[1]);
    }
    break;
case 3:

    printf("\nTurno del jugador %d\n", jugador[0].turno?1:2);
    printf("\n\tMueve la trayectoria del disparo con las flechas de arriba y abajo, o con 'w' o 's'\n");

    if(jugador[0].turno){
        updateTablero(&jugador[0], &jugador[1], tablero);
        inputAim(&jugador[0], &jugador[1], tablero, &keyMain, &c2, &potencia, &moment);
        printf("\n\tObjetivos restantes: %d\n\n", soldadosActivos(&jugador[1]));
    } else {
        updateTablero(&jugador[1], &jugador[0], tablero);
        inputAim(&jugador[1], &jugador[0], tablero, &keyMain, &c2, &potencia, &moment);
        printf("\n\tObjetivos restantes: %d\n\n", soldadosActivos(&jugador[0]));
    }

    break;

case 4:

    //Imprime cual de los jugadores gano la partida.
    system("cls");
    inicializarTablero(tablero); updateFinal(&jugador[0], tablero); updateFinal(&jugador[1], tablero);
    imprimirTableroAux(tablero);

    printf("\n\n=====\\n\n");
    if(soldadosActivos(&jugador[0]) == soldadosActivos(&jugador[1])){
        printf("Ha sido un empate");
    } else {
        printf("El jugador %d ha ganado.", (soldadosActivos(&jugador[0]) > soldadosActivos(&jugador[1]))? 1: 2);
    }

    printf("\n\n=====\\n\n");

    moment++;
    break;

```

```

    }

    if(keyMain == 27){
        desplegarPausa(&keyMain, tablero, jugador, &moment, &c1, &c2);
    }

    sleep(10);
} while (keyMain != 27 && moment<=4); // '27' es el codigo ASCII para la tecla 'Escape'

if(keyMain!=27){
    printf("Presione una tecla para ver el feedback del juego.");
    getch();
    mostrarFeedback(&jugador[0], &jugador[1]);
    system("pause");
}

return 0;
}

```

RF-012: Interacción multijugador local.

Este requerimiento se cumple desde el primer momento en el que se ejecuta la función de “iniciarjuego” mostrada anteriormente. Desde dicha función se hace la ejecución de la partida que permite las interacciones entre los jugadores.

Observación: Los requerimientos funcionales del 009 – 011 no se incluyen en este reporte debido a que al final no se implementaron en la codificación del proyecto. Esto era algo que ya teníamos considerado omitir si no había suficiente tiempo.

RF-013: Tablero de juego.

La implementación de este requerimiento funcional en el sistema se llevó a cabo como estaba planificada, se puede observar que la función llamada “imprimirTablero” permite visualizar el tablero del juego durante la partida. A su vez

```
//Funciones para la impresi?n del tablero de juego. //SALIDAS
void imprimirTableroAux(char tablero[FILAS][COLUMNAS])
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbiInfo; //Funciones para la actualizaci?n del buffer de la consola.
    GetConsoleScreenBufferInfo(hConsole, &csbiInfo);

    for (i = 0; i < FILAS; i++) {
        COORD pos = {0, i};
        SetConsoleCursorPosition(hConsole, pos);
        for (j = 0; j < COLUMNAS; j++) {
            printf("%c", tablero[i][j]);
        }
    }
}

//FUNCIONES PARA EL PROCESO DEL VIDEOJUEGO (opción "jugar" en el menú principal).
void inicializarTablero(char tablero[FILAS][COLUMNAS]) //PROCESO //Inicia los valores guardados en la matriz que dibujara el tablero.
{
    for (i = 0; i < FILAS; i++) {
        for (j = 0; j < COLUMNAS; j++) {
            if (i == 0 || i == FILAS - 1) {
                tablero[i][j] = '-';
            } else if (j == 0 || j == COLUMNAS - 1) {
                tablero[i][j] = '|';
            } else if (j == (COLUMNAS/2)-1 || j == (COLUMNAS/2)+1){
                tablero[i][j] = '|';
            } else {
                tablero[i][j] = ' ';
            }
        }
    }
}
```

RF-014: Colocar tropas.

A continuación, se presentan imágenes de la función principal que se encarga de colocar las tropas es “moveSoldier”. Esta función permite mover las tropas del jugador actual para luego colocarlas, así como no permitir seleccionar lugares ocupados o fuera del tablero. La implementación del requerimiento fue como se había planificado, incluso agregando un macro para cambiar el número máximo de soldados que se permite por jugador, siendo 12 el número por defecto.

```

2 void moveSoldier(player* jugador, int direccion, char tablero[FILAS][COLUMNAS], int* c) //Funcion para controlar la posicion donde se colocan soldados.
3 {
4     int nuevaPosX = jugador->soldier[*c].posX;
5     int nuevaPosY = jugador->soldier[*c].posY;
6
7     switch(direccion){
8         case 72: case 119: case 87: // Flecha arriba, w y W
9             if(tablero[nuevaPosY-1][nuevaPosX] == ' '){ //Analiza si la casilla es valida para moverse y colocar un soldado.
10                 nuevaPosY--;
11             }
12             break;
13         case 80: case 115: case 83: // Flecha abajo, s y S
14             if(tablero[nuevaPosY+1][nuevaPosX] == ' '){
15                 nuevaPosY++;
16             }
17             break;
18         case 75: case 97: case 65: // Flecha izquierda, a y A
19             if(tablero[nuevaPosY][nuevaPosX-1] == ' '){
20                 nuevaPosX--;
21             }
22             break;
23         case 77: case 100: case 68: // Flecha derecha, d y D
24             if(tablero[nuevaPosY][nuevaPosX+1] == ' '){
25                 nuevaPosX++;
26             }
27             break;
28         case 13: //Tecla enter

```



```

        break;
    case 13: //Tecla enter

        tablero[jugador->soldier[*c].posY][jugador->soldier[*c].posX] = '#'; //Imprime un "sprite" del soldado.

        //Guarda las posiciones donde el jugador en turno guarda sus soldados.
        if (tablero[jugador->soldier[*c].posY][jugador->soldier[*c].posX + 1] == ' '){
            jugador->soldier[*c + 1].posX = jugador->soldier[*c].posX + 1;
            jugador->soldier[*c + 1].posY = jugador->soldier[*c].posY;
        } else if (tablero[jugador->soldier[*c].posY][jugador->soldier[*c].posX - 1] == ' '){
            jugador->soldier[*c + 1].posX = jugador->soldier[*c].posX - 1;
            jugador->soldier[*c + 1].posY = jugador->soldier[*c].posY;
        } else if (tablero[jugador->soldier[*c].posY - 1][jugador->soldier[*c].posX] == ' '){
            jugador->soldier[*c + 1].posX = jugador->soldier[*c].posX;
            jugador->soldier[*c + 1].posY = jugador->soldier[*c].posY - 1;
        } else if (tablero[jugador->soldier[*c].posY + 1][jugador->soldier[*c].posX] == ' '){
            jugador->soldier[*c + 1].posX = jugador->soldier[*c].posX;
            jugador->soldier[*c + 1].posY = jugador->soldier[*c].posY + 1;
        } else {
            jugador->soldier[*c + 1].posX = jugador->canon.posX;
            jugador->soldier[*c + 1].posY = jugador->canon.posY - 2;
        }

        break;
    default:
        return; // No se reconoce la tecla, no hacer nada
}

```

RF-015: Potencia de proyectiles.

El requerimiento se implementó tal como se planificó, estableciendo potencias entre las que puede elegir el usuario mediante números que van del 1–11. La función principal que se encarga de la potencia de los proyectiles es “determinarPotencia”. Esta función permite al jugador elegir la potencia de disparo.

```
void determinarPotencia(int* potencia, player* jugador, int c, char* key, char tablero[FILAS][COLUMNAS]) //Funcion para calcular la potencia de tiro.
{
    *key=72;
    system("cls");
    do{
        imprimirTableroAux(tablero);

        printf("\nSeleccione la potencia:\n\n\t");
        if (*potencia == 1) {
            printf("< %d > ", *potencia);
        } else {
            printf("%d... < %d > ", *potencia - 1, *potencia);
        }
        fflush(stdout);
        if(*potencia != POTENCIA_MAX){
            printf("...%d", *potencia + 1);
        }
        printf("\n\n\tPara seleccionar la potencia, use las flechas izquierda y derecha, o las teclas 'A' y 'D'");
    }
}
```

```
if(kbhit()){ //Controlar el movimiento con las fechas o WSAD
    *key = getch();
    switch(*key){
        case 75: case 97: case 65: // Flecha izquierda, a y A
            if (*potencia > 1){
                (*potencia)--;
            }
            break;
        case 77: case 100: case 68: // Flecha derecha, d y D
            if (*potencia < POTENCIA_MAX){
                (*potencia)++;
            }
            break;
    }
} while (*key != 13 && *key != 27);
system("cls");
}
```

Función que permite elegir la potencia del proyectil, así como implementarla.

RF-016: Trayectoria de proyectiles.

La implementación del requerimiento fue conforme a lo planificado, sin embargo, cabe explicar que la implementación del cálculo fue diferente a nuestra idea principal, ya que al estar realizándolo en consola se tuvo que calcular la trayectoria estableciendo coordenadas en el tablero de juego, luego por medio de algebra calcular el pendiente de acuerdo a la posición del cañón del jugador que este lanzando en ese momento, considerando igual la dirección y la potencia con la que se lanza el proyectil.

La función principal que se encarga de la trayectoria de los proyectiles es “disparar”. Esta función visualiza la trayectoria calculada atreves de la dirección dada y de la potencia cargada, así como encargarse que sea visualizada una animación del disparo.

```
//Funciones para los procesos durante la ejecucion de una partida.
int f(int x, float m, int origen) //Función para calcular la trayectoria del proyectil.
{
    //m es la pendiente.
    //x es el valor donde esta ubicado el tablero.
    //origen: sirve para determinar el valor de x, dependiendo del lado de tablero de donde sale la trayectoria.

    float fx;
    x-=origen;

    if(origen < COLUMNAS/2){
        fx= ((FILAS/2-1) - (float) m*x);
    } else {
        fx= ((FILAS/2-1) + (float) m*x);
    }

    if(m>=0){
        return floor(fx);
    } else {
        return ceil(fx);
    }
}
```

```

void disparar(player* jugador, player* enemigo, int c, int potencia, char tablero[FILAS][COLUMNAS]) //Funcion para controlar el comportamiento de un disparo
{
    int posX, posY, direccion, cont;
    char key;

    if(jugador->canon.posX < COLUMNAS/2){
        posX=COLUMNAS/2 + 1 + (potencia * 4); //Se toma en cuenta el valor almacenado en la potencia
        direccion = 1;
    } else {
        posX=COLUMNAS/2 + 2 - (potencia * 4);
        direccion = -1;
    }

    if(posX<1){
        posX=1;
    } else if (posX>COLUMNAS-2){
        posX = COLUMNAS-2;
    }

    for(i = posX; i > (posX - MAX_DAMAGE); i--){
        posY = f(i, (float) c/MAX_PTR, jugador->canon.posX);

        for(j = 0; j<MAX_SOLDIER; j++){
            if((posY == enemigo->soldier[j].posY || posY - 1 == enemigo->soldier[j].posY || posY + 1 == enemigo->soldier[j].posY) && i == enemigo->soldier[j].posX ){
                enemigo->soldier[j].activo = 0;
            }
        }
    }
}

```

```

    }
}

// animacion de disparo;

cont = jugador->canon.posX;

for(i=0; i<MAX_AIM; i++){
    tablero[jugador->aim[i].posY][jugador->aim[i].posX] = ' ';
}

if(jugador->canon.posX < COLUMNAS/2){
    do {
        posY = f(cont, (float) c/MAX_PTR, jugador->canon.posX);

        tablero[posY][cont] = '*';

        imprimirTableroAux(tablero);
        printf("\n\nPresiona enter para omitir.");

        Sleep(((int)10*POTENCIA_MAX/potencia);

        tablero[posY][cont] = ' ';

        cont += direccion;

        verifVacio(tablero, posY, &cont, direccion, (float)c/MAX_PTR, jugador->canon.posX);
        if(kbhit()){
            key = getch();
            if (key==13){
                break;
            }
        }
    }
}

} while (cont < posX);
} else {
    do {
        posY = f(cont, (float) c/MAX_PTR, jugador->canon.posX);

        tablero[posY][cont] = '*';

        imprimirTableroAux(tablero);
        printf("\n\nPresiona enter para omitir.");

        Sleep(((int)10*POTENCIA_MAX/potencia);

        tablero[posY][cont] = ' ';

        cont += direccion;
        verifVacio(tablero, posY, &cont, direccion, (float)c/MAX_PTR, jugador->canon.posX);
        if(kbhit()){
            key = getch();
            if (key==13){
                break;
            }
        }
    }
} while (cont > posX && f(cont, (float) c/MAX_PTR, jugador->canon.posX) < FILAS-1 && f(cont, (float) c/MAX_PTR, jugador->canon.posX) > 0 && cont > 0 && cont < COLUMNAS-1);
}

```

RF-017: Turnos de jugadores.

Este requerimiento se implementó de forma exitosa. La función que se debe detectar cuando se debe hacer el cambio de turno entre los dos jugadores es “switchTurno”. Se presenta una imagen de esta última.

```
void switchTurno ( player* jugador1, player* jugador2) //Funcion para el cambio de turno.
{
    jugador1->turno = (jugador1->turno + 1) % 2;
    jugador2->turno = (jugador2->turno + 1) % 2;
}
```

RF-018: Daño de área.

La función que se encarga del daño en área, así como visualizarlo, es “disparar”. Esta función al encargarse del disparo implementa el daño en área, así como la visualización de este por medio de la impresión de símbolos en la pantalla del usuario.

```
for(i = posX - 1; i > (posX - MAX_DAMAGE + 1); i--){ //Deteccion de las colisiones
    posY = f(i, (float) c/MAX_PTR, jugador->canon.posX);

    if(tablero[posY][i] == ' '){
        tablero[posY][i] = '0';
    } else if (tablero[posY][i] == '#'){
        tablero[posY][i] = 'X';
    }
}

imprimirTableroAux(tablero); //Impresion de la colisi?n
Sleep(500);
for(i = posX; i > (posX - MAX_DAMAGE); i--){
    posY = f(i, (float) c/MAX_PTR, jugador->canon.posX);

    for(j=posY-1; j<posY+2; j++){
        if(tablero[j][i] == ' '){
            tablero[j][i] = '0';
        } else if (tablero[j][i] == '#'){
            tablero[j][i] = 'X';
        }
    }
}

imprimirTableroAux(tablero);
}
```

RF-019: Detección de proyectiles.

La función que se encarga de verificar la colisión, así como de visualizarla, es “disparar”.

Esta función al encargarse del disparo implementa el daño en área, que, a su vez, verifica las colisiones, así como su visualización.

RF-020: Determinar ganador.

La función encargada de validar si alguien ha ganado y proclamarlo como el ganador es "iniciarJuego". Esta función, cada vez que se realiza un disparo, verifica si hay algún ganador. En caso afirmativo, finaliza el juego al término de cada ronda y determina al ganador o si hay empate.

RF-021: FeedBack.

La función responsable de proporcionar el feedback de la partida es "iniciarJuego".

Después de que una partida ha finalizado, esta función invoca a la función “mostrarFeedback” e imprime los disparos realizados, las tropas del enemigo eliminadas y las tropas restantes de cada jugador.

```
//SALIDAS
void mostrarFeedback(player* jugador1, player* jugador2)//Imprime en la pantalla una retroalimentación para los jugadores.
{
    printf("\n\n=====\\n");
    printf("      ESTADISTICAS JUGADOR 1\\n\\n");
    printf("El jugador 1 realizo %d disparos\\n", jugador1->disparos);
    printf("El jugador 1 elimino a %d soldados enemigos\\n", (MAX_SOLDIER - soldadosActivos(jugador2)));
    printf("El jugador 1 se quedo con %d soldados\\n\\n", soldadosActivos(jugador1));
    printf("=====\\n");
    printf("      ESTADISTICAS JUGADOR 2\\n\\n");
    printf("El jugador 2 realizo %d disparos\\n", jugador2->disparos);
    printf("El jugador 2 elimino a %d soldados enemigos\\n", (MAX_SOLDIER - soldadosActivos(jugador1)));
    printf("El jugador 2 se quedo con %d soldados\\n\\n", soldadosActivos(jugador2));
}
```

RF-022: Pausar la partida.

La función encargada de manejar el menú de pausa se denomina "desplegarPausa". Esta función es invocada desde "iniciarJuego" cuando se detecta que el usuario ha presionado la tecla "esc". El menú despliega tres opciones: reanudar partida, que retorna al juego normal; reiniciar partida, que restablece todos los movimientos de ambos jugadores, permitiendo comenzar desde el principio; y volver al menú principal, que reinicia la partida y te devuelve al menú principal.

```
void desplegarPausa(char* keyMain, char tablero[FILAS][COLUMNAS], player jugador[], int* moment, int* c1, int* c2)//Funcion para pausar la partida.
{
    char reanudar[] = "> Reanudar partida", reiniciar[] = "Reiniciar partida", salir[] = "Salir del juego", opcion;
    int opc = 1;

    system("cls");

    do{
        imprimirTableroAux(tablero);
        printf("\n\n===== \n");
        printf("                PAUSA\n\n");
        printf("                %sReanudar partida\n",opc==1?"> " : " ");
        printf("                %sReiniciar partida\n",opc==2?"> " : " ");
        printf("                %sRegresar al Menu\n",opc==3?"> " : " ");
        if(kbhit()){
            opcion = getch();
            switch(opcion){
                case 72: case 119: case 87: // Flecha arriba, w y W.
                    if(opc>1 ){
                        opc--;
                    }
                    break;

                case 80: case 115: case 83: // Flecha abajo, s y S.
                    if(opc<3){
                        opc++;
                    }
                    break;

                case 13: // enter
            }
        }
    } while(1);
}
```

```

        printf("\nReanudando partida...");
        Sleep(500);
        system("cls");
        break;
    case 2:
        printf("\nReiniciando partida...");
        Sleep(500);

        *keyMain = 1;
        *moment = 1;
        *c1=0;
        *c2=0;

        jugador[0].turno=1;
        jugador[1].turno=0;
        jugador[0].disparos=0;
        jugador[1].disparos=0;

        inicializarTablero(tablero);
        setCanon(jugador, tablero);
        setSoldier1(&jugador[0], tablero); setSoldier1(&jugador[1], tablero);
        system("cls");

        break;
    case 3:
        // HACER NADA.
        printf("\nVolviendo al menu...");
        Sleep(500);
        break;
    }
    break;
}
}
} while (opcion!=13);
}

```

La función encargada de manejar el menú de pausa se denomina "desplegarPausa". Esta función es invocada desde "iniciarJuego" cuando se detecta que el usuario ha presionado la tecla "esc". El menú despliega tres opciones: reanudar partida, que retorna al juego normal; reiniciar partida, que restablece todos los movimientos de ambos jugadores, permitiendo comenzar desde el principio; y volver al menú principal, que reinicia la partida y te devuelve al menú principal.

Acceso al código completo

https://github.com/JarethJaziel/Abyss_Battle/tree/Code

15.0 Reporte de evaluación basada en objetivos

Integrante	Participaciones individuales	Evidencias	Aportación total para el equipo
Alonzo Palacios Rodrigo	<ul style="list-style-type: none"> • Elaboración de la presentación de los avances del proyecto • Codificación • Descripción de requerimientos y casos de uso • Revisión y corrección del archivo anterior Redacción en el archivo	<ul style="list-style-type: none"> • Archivo de reporte de resultados. • Presentación del reporte de resultados. • Código “main” del programa. 	100%
Cuevas García Braulio Samuel	<ul style="list-style-type: none"> • Elaboración de la presentación de los avances del proyecto • Codificación • Descripción de requerimientos y casos de uso • Revisión y corrección del archivo anterior Redacción en el archivo	<ul style="list-style-type: none"> • Archivo de reporte de resultados. • Presentación del reporte de resultados. • Código “main” del programa. 	100%
Martínez Martínez José Pablo	<ul style="list-style-type: none"> • Elaboración de la presentación de los avances del proyecto • Codificación • Descripción de requerimientos y casos de uso • Revisión y corrección del archivo anterior Redacción en el archivo	<ul style="list-style-type: none"> • Archivo de reporte de resultados. • Presentación del reporte de resultados. • Código “main” del programa. 	100%
Moo Pan Jareth Jaziel			100%

	<ul style="list-style-type: none">• Elaboración de la presentación de los avances del proyecto• Codificación• Descripción de requerimientos y casos de uso• Revisión y corrección del archivo anterior Redacción en el archivo	<ul style="list-style-type: none">• Archivo de reporte de resultados• Presentación del reporte de resultados• Código main del programa	
--	---	--	--

Referencias

Comments (GNU Coding Standards). (s. f.).

https://www.gnu.org/prep/standards/html_node/Comments.html

El sistema operativo GNU y el movimiento del software libre. (s. f.).

<https://www.gnu.org/home.es.html>

GNU Coding Standards - GNU Project - Free Software Foundation. (s. f.).

<https://www.gnu.org/prep/standards/>

<https://www.platoapp.com>

Information for Maintainers of GNU Software. (s. f.).

<https://www.gnu.org/prep/maintain/maintain.html>

Names (GNU Coding Standards). (s. f.).

https://www.gnu.org/prep/standards/html_node/Names.html

Plato. (s. f.). <https://www.platoapp.com/>

Sea Battle 2 - Apps en Google Play. (s. f.).

https://play.google.com/store/apps/details?id=com.byril.seabattle2&hl=es_MX

Alonzo, Cuevas, Martínez, Moo

Introducción

El proyecto de software busca reinventar juegos de mesa clásicos en formato de videojuego y se eligió un juego 1v1 basado en "Battle Ship", con temática medieval, programado en C. "Abyss Battle" se destaca por su mecánica de lanzamiento de proyectiles mediante cañones, una característica única en juegos de batallas navales. Es importante saber que se busca simular un juego 2D que busque una batalla entre 2 jugadores, con una mecánica de precisión en sus proyectiles con un software desarrollado.

Metodología

El proyecto se dividió en 2 partes: La documentación y el código ejecutable.

Documentación:

Se dividió a su vez en 2 etapas: En la primera etapa se busca una recopilación exhaustiva de de objetivos específicos del software, casos de uso, definición de estándar de codificación y tipos de usuario, además de tener bastante claro la descripción del software.

En la segunda etapa se espera una recopilación muy específica de los requerimientos funcionales y no funcionales, cumpliendo con los módulos de los objetivos específicos acordados en la fase anterior.

Codificación:

Dividido en 2 partes: La primera parte se buscaba recopilar información de bibliotecas, funciones e interfaces que se puedan implementar para la optimización del ejecutable, el cual se tomó 2 semanas en realizarse, seguidamente se inició el código ejecutable para luego obtener la versión finalizada del software.

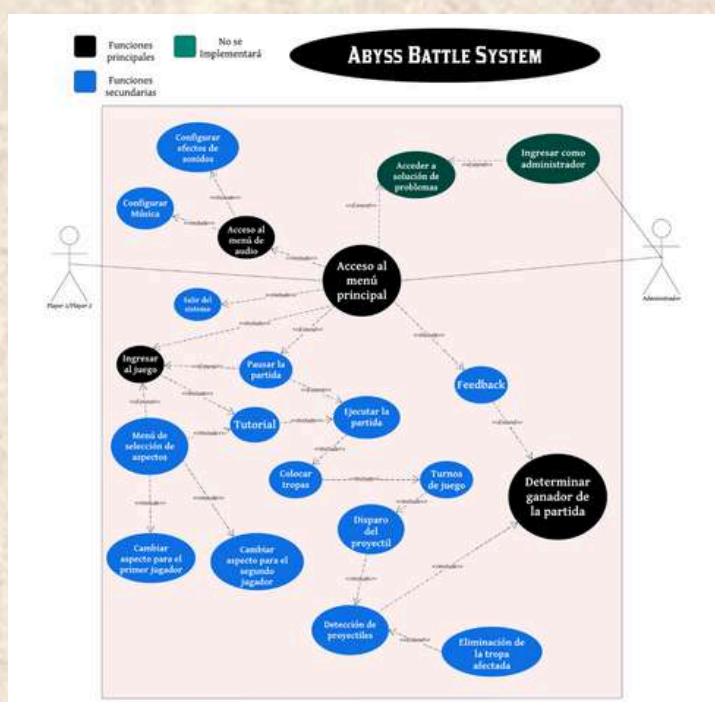


Diagrama de casos de uso en el cual se busca cumplir en el código

Resultados

Finalmente se realizó un código ejecutablem cumpliendo el objetivo de desarrollar el juego 2D 1v1 con mecánica de proyectiles precisos, en el cual cumplió con lo establecido en la documentación

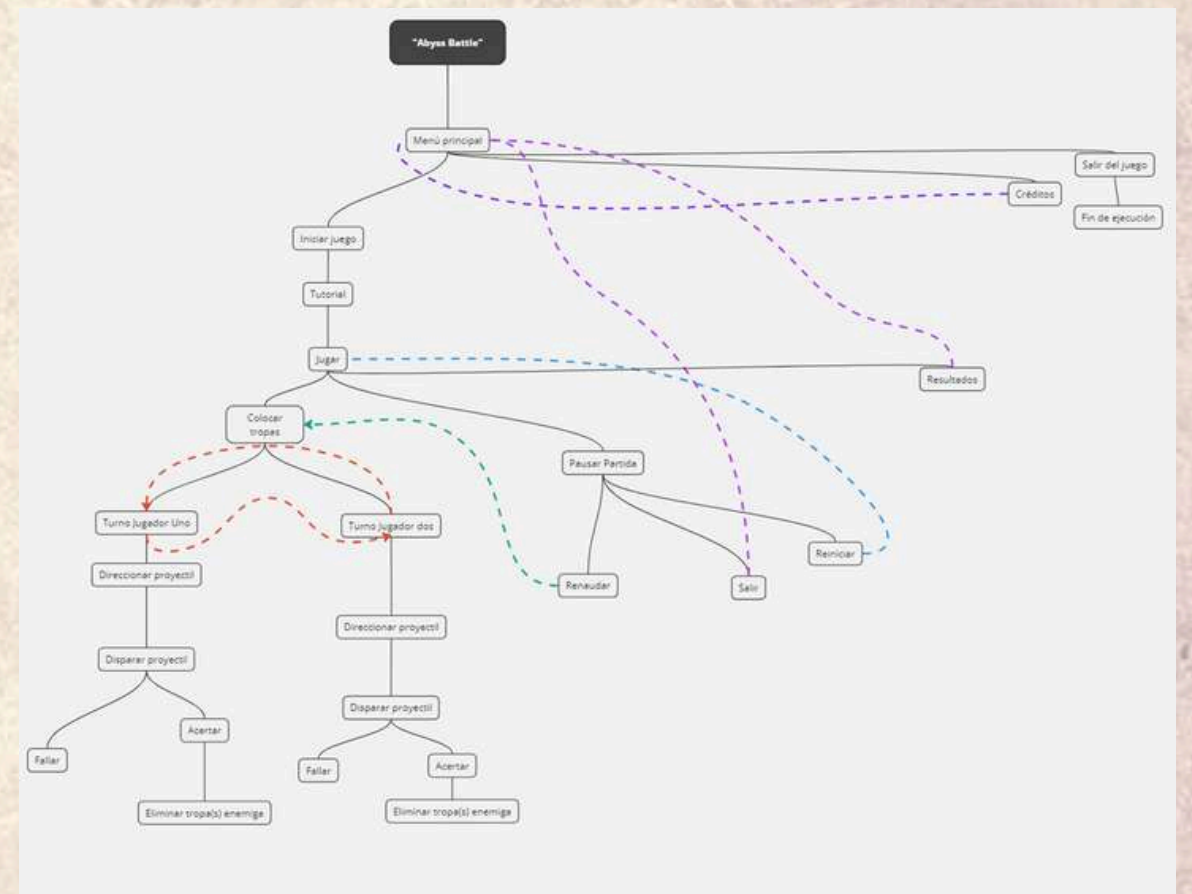
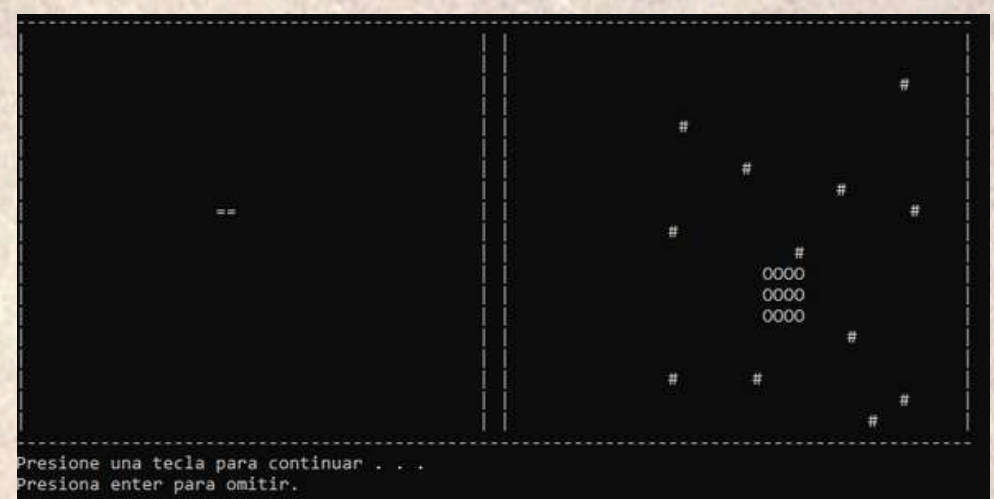


Diagrama a bloques en el que se especifica las diferentes opciones dentro del software

Conclusiones

El enfoque de este proyecto, centrado en la mecánica innovadora de lanzamiento de proyectiles mediante cañones, ofrece una propuesta única y distintiva en el de innovación sobre los demás proyectos realizados, logrando exitosamente un ejecutable en lenguaje C capaz de detectar una partida 1v1 en el cual se aplican trayectorias y fuerzas de los cañones. Aunque otros juegos presentan características similares en términos de jugabilidad y modalidades de juego, la singularidad de "Abyss Battle" radica en su enfoque estructurado en el lenguaje C y su orientación hacia el juego local en computadoras.



Software ejecutable “Abyss Battle”