**NARRATIVA**

**Introducción**

Este proyecto, titulado "Battle Ship" o mayormente conocido como "Batalla Naval", fue concebido con el propósito de proporcionar una alternativa única, de carácter estratégico y divertida de entretenimiento, alejándose de las opciones convencionales.

**¿En qué consiste?**

Este mismo, consiste en un enfrentamiento entre dos participantes, donde cada uno dispone de sus respectivas naves de guerra. Los jugadores se atacan mutuamente a través de una secuencia de turnos, intentando deducir la ubicación de las naves del adversario con el objetivo de "hundirlas". El vencedor será el participante que consiga hundir la totalidad de la flota enemiga en primer término.

**Preparación**

Al iniciar el juego, se establecerán las naves de guerra en lugares totalmente aleatorios con el objetivo de otorgar una mayor dificultad para el jugador y que posteriormente resulte mucho más desafiante.

**Diario de trabajo**

**Día – 16 / 05**

**Cosas Hechas en el Día**

Hoy nos pudimos juntar con el grupo entero y discutir sobre cómo será el trabajo final.

* Pudimos concluir el objetivo del trabajo Batalla Naval.
* Pensar en las Struct que usaremos.
* Hablamos sobre cómo será el diseño.
* Las posibles bibliotecas que crearemos.
* Y la estructura de trabajo que tendremos.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 17 / 05**

**Cosas Hechas en el Día**

Logramos terminar la idea y empezar con el proyecto.

Aprendimos a utilizar las Bibliotecas y unirlas con los Archivos.

Ian Elaboro toda la parte del título.

* Texto Grande
* Recuadros
* Gotoxy

**Problemas Encontrados**

No sabemos para qué utilizar el segundo Archivo para el proyecto.

**Solución**

Encontramos como posible solución agregar un archivo con los niveles de dificultad, y además otro para las reglas del juego.

**Día – 18 / 05**

**Cosas Hechas en el Día**

Comenzamos la librería de cuentas de usuarios, la cual abarca los siguientes temas:

* [x] Crear cuenta (Única completada)
* [] Iniciar sesión
* [] Modificar usuario
* [] Cerrar sesión
* [] Dar de baja

**Problemas Encontrados**

La función principal quedaba muy extensa, por lo que tuvimos que buscar una alternativa. Sin embargo, terminamos modularizando excesivamente.

**Solución**

Encontramos como posible solución agregar un archivo con los niveles de dificultad, y además otro para las reglas del juego.

**Día – 19 / 05**

**Cosas Hechas en el Día**

Hoy continuamos con el desarrollo de la librería de cuentas.

Pudimos completar el apartado de **“Iniciar Sesión”.**

Las funciones utilizadas fueron 2 en total:

* **Iniciar Sesión,** encargada de leer nombre de usuario y contraseña, además de llamar a la función de comprobación. Retorna el struct del usuario.
* **Comprobación iniciar sesión,** la cual recibe el nombre de usuario y la contraseña, comprobando que ambas existan y pertenezcan al mismo usuario. Retorna un flag.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 20 / 05**

**Cosas Hechas en el Día**

Hicimos las preguntas que teníamos

* ¿Borrar o dar de baja? → Dar de Baja.
* 3 archivos
* Usuarios
* Dificultad
* Reglas
* Usar los Arreglos para ordenar.

**Problemas Encontrados**

1. No inicializa las demás partes de la estructura (estadísticas, estado).

2. Al iniciar Sesión, solo está pasando nombre y contraseña, no pasa todo lo demás.

**Solución**

1. Ya se corrigió, inicializando al momento de crear cuenta, las estadísticas en 0 y el estado del usuario en 1.

2. Pasamos el struct del usuario como puntero.

**Día – 21 / 05**

**Cosas Hechas en el Día**

Corregí lo hecho hasta el momento, modularizando en las partes necesarias y eliminando código sobrante.

Realicé de la parte de Opciones del Menú principal toda la sección de Modificar Usuario, la cual da la posibilidad al respectivo usuario de cambiar su nombre y/o contraseña.

**Problemas Encontrados**

Al trabajar directamente con la estructura del usuario en lugar del archivo completo, encontré un inconveniente al intentar sobrescribir los datos del usuario en el archivo durante la sección de Modificar Usuario.

Esto resultó así ya que no contaba con la posición exacta de la estructura del usuario en el archivo, lo que resultaba en la modificación de los datos de otro usuario por error.

**Solución**

Como solución, incorporé una nueva función:

* **Buscar posición en archivo:** Se encarga de buscar la posición exacta de la estructura del usuario en el archivo y retornándola.

Esta solución permite guardar los datos a sobrescribir de manera precisa en el archivo, asegurando que se modifiquen los datos del usuario correcto.

**Día – 22 / 05**

**Cosas Hechas en el Día**

**NULL**

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 23 / 05**

**Cosas Hechas en el Día**

Hoy empecé con el desarrollo del juego.

Comencé creando la tabla del mismo.

- [x] Tabla con Matrices

- [ ] Agregar Barcos

**Problemas Encontrados**

Logré crear una versión funcional, pero no permite ser reutilizado, teniendo que crear 3 funciones para cada dificultad.

Lamentablemente, al testear por segunda vez, descubrí que, por alguna razón, las dificultades medio y difícil no las arma. El error parece encontrarse en el momento de llenar la matriz con ceros.

**Solución**

Creé una función con la capacidad de desarrollar las 3 dimensiones de la tabla *(5x5, 10x10 y 15x15),* se le pasa por parámetro desde el main, la dimensión de la tabla y la función se encarga de dibujarlo con esas medidas.

**Día – 26 / 05**

**Cosas Hechas en el Día**

Hoy me concentré completamente en el desarrollo del juego.

El día anterior (23/05), logré hacer que con una matriz se genere la tabla del juego.

La misma es una función que se le pasa la dimensión deseada:

* 5 x 5 (Fácil)
* 10 x 10 (Medio)
* 15 x 15 (Difícil)

Lo que hice hoy fue modularizar la función y mejorarla.

La misma tenía problemas, como, por ejemplo, no generaba las tablas del nivel Medio y Difícil, y tampoco permitía agregarle los barcos.

Ahora logré que los barcos puedan agregarse y verse en la tabla (mostrarTablaUsuario), representando el 0 como Agua y el 1 como Barcos.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 28 / 05**

**Cosas Hechas en el Día**

Mejoramos la función de Agregar barcos de la siguiente manera:

* Modularíamos la función en 3 secciones:
* Agregar barcos: Se encarga de llamar a las demás funciones y agrega los barcos a la tabla.
* Pedir coordenadas barcos: Se ocupa tanto de pedir las coordenadas del barco y su dirección *(horizontal o vertical),* como también de comprobar que la dirección sea válida.
* Comprobación barcos: Se encarga de comprobar que los barcos no se superpongan entre sí, ni que pasen los límites del tablero.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 29 / 05**

**Cosas Hechas en el Día**

Mejoramos el apartado ***Comprobación barcos*,** dividiéndola en **2 funciones** diferentes:

* **Comprobación superposición barcos**, que lleva a cabo la tarea de comprobar que las coordenadas ingresadas no se superpongan con otros barcos, ni que el mismo no supere los límites de la tabla.
* **Comprobación límite barcos**, que se encarga de comprobar que las coordenadas ingresadas no superen los límites del tablero.

Optamos por dividir la función de esta manera para **proporcionar dos mensajes distintos en caso de que se ingresen coordenadas inválidas:** uno para *cuando los barcos se superponen* y otro para *cuando se exceden los límites del tablero.*

También trabajamos en el apartado de la creación del ***tablero del rival (BOT)***. Así, creamos **2 funciones:**

* **Crear rival BOT**, la cual llama a la otra función la cantidad de veces necesaria, pasándole por parámetro la longitud del barco, la dimensión del tablero y el tablero. Crea el tablero del rival.
* **Agregar barcos rival**, que se ocupa de agregar los barcos de manera aleatoria en el tablero. Llama a las funciones de comprobación anteriormente creadas.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 30 / 05**

**Cosas Hechas en el Día**

En el transcurso de hoy, nos enfocamos en mejorar y corregir el código, eliminando cualquier parte innecesaria. Además, **dimos inicio al desarrollo de la jugabilidad**, creando las siguientes funciones:

* **Jugar batalla naval**, que llama a las demás funciones y trabaja con un ciclo infinito hasta que un jugador gane.
* **Mostrar tabla jugando**, la cual muestra la tabla del rival, actualizándose constantemente a medida que el usuario ataca *(no muestra donde se esconden sus barcos)*.
* **Disparo jugador**, que llama a la función de pedir coordenadas. También llama a la función de comprobación, da sus respectivos mensajes y cambia los valores de la tabla *(a agua impactada o barco impactado)*.
* **Pedir coordenadas disparo.**
* **Comprobación disparo superposición**, que se encarga de comprobar que la coordenada ingresada no haya sido atacada previamente por el usuario.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 31 / 05**

**Cosas Hechas en el Día**

**Los avances del día fueron:**

* Decidir qué **signos** usaremos para las tablas del Batalla Naval.

|  |  |  |
| --- | --- | --- |
| **Valor en la tabla** | **Código ASCII** | **Función o Acción** |
| -2 | 158 | (Impacto Agua) |
| -1 | 157 | (Impacto Barco) |
| 0 | 254 | (Agua) |
| 1 | 219 | (Barcos) |

* **Preguntar las dudas que teníamos** y ver cosas para cambiar.
* **Avanzar con el apartado de Jugabilidad**, creando un nuevo ***“MostrarTablaJugando”*** y las primeras funciones para pedir las coordenadas del disparo.
* Reducir la dimensión de la tabla en el **Modo Difícil a 13x13**

**Problemas Encontrados**

Con esas preguntas logramos ver *“problemas”* o *“cosas”* para cambiar y son:

* Encontrar otra manera de inicializar las tablas.
* *Anteriormente, las Tablas eran creadas con la dimensión de 15x15, causando que guarde espacio innecesario en la memoria, debemos buscar una manera de crearlas en la dimensión que pida el usuario.*
* Ver dónde utilizar la recursividad.
* *Íbamos a utilizarla para crear los turnos en la partida, pero la función en si consume mucha memoria y puede fallar en dispositivos de menor Gama.*

**Solución**

**NULL**

**Día – 04 / 06**

**Cosas Hechas en el Día**

Hoy mejoré la sección de Dar de baja, con el objetivo de incorporar un *captcha* para que el proceso de invalidar un usuario sea *más seguro*.

Para lograr esto generé la siguiente función:

* *Comprobación dar de baja*, la cual crea un número de 1000 a 9999 de manera aleatoria. El usuario debe ingresar el número correctamente para que su cuenta se dé de baja con éxito.

Además, creé una nueva función en la librería de cuentas:

* *Recuperar cuenta*, en la cual el usuario debe ingresar su nombre de usuario y contraseña correctamente para recuperar su cuenta dada de baja anteriormente. Esta función aparece en la pantalla principal, junto a ‘Iniciar sesión’, ‘Crear cuenta’ y ‘Salir del juego’.

**Problemas Encontrados**

**NULL**

**Solución**

**NULL**

**Día – 14 / 06**

**Cosas Hechas en el Día**

**Retomamos el proyecto después de parciales.**

Repasamos todo el código para **buscar errores** y **cosas para modificar.**

Los cambios que hicimos fueron:

* **Modificar** Iniciar Sesión.
* **Modificar** Dar de Baja y Recuperar Cuenta.
* **Comenzar** la función de comprobación de Barcos Hundidos
* **La implementación de Pilas** como *“barra de vida”* en el juego.

**Problemas Encontrados**

Uno de los problemas encontrados fue la recursividad accidental que se generaba al momento de manejarse entre los distintos menús.

Esto generaba un conflicto al intentar cerrar la sesión y finalizar el programa.

**Solución**

Para solucionarlos, tuvimos que eliminar las llamadas relacionadas a los menús, en el código, evitando la recursividad.

Y para lograr cerrar sesión desde el menú de opciones, sin pasar por el Menú Principal, creamos un puntero que mande directo a cerrar sesión.

**Día – 15 / 06**

**Cosas Hechas en el Día**

Empezamos con el desarrollo del apartado *“Otros Usuarios”*, la cual tiene como finalidad *buscar un jugador* específico (a través del nombre) y ver *lista de jugadores*.

Las funciones creadas fueron:

* Otros usuarios: se elige entre buscar jugador o ver lista de jugadores. Llama a las demás funciones.
* Pasar usuarios a arreglo: crea arreglo dinámico para pasar los registros del archivo. Llama a las siguientes funciones:
* Contar registros del archivo: cuenta la cantidad total de registros en el archivo para crear arreglo dinámico.
* Copiar usuarios: pasa los usuarios del archivo al arreglo.
* Ver jugador: usuario ingresa nombre de jugador que quiere buscar. Muestra registro de jugador ingresado o muestra mensaje de error si no existe. Llama la siguiente función:
* Buscar jugador: busca jugador ingresado en el arreglo. Si lo encuentra guarda posición del registro y retorna 1, sino retorna 0.
* Lista de jugadores: se elige entre ver lista de jugadores ordenado por orden alfabético o por puntaje. Llama a las siguientes funciones:
* Ordenación selección orden alfabético: ordena (a través de método de selección) el arreglo por orden alfabético.
* Encontrar menor: busca la posición del menor dato del arreglo (por nombre) y la retorna.
* Ordenación inserción por puntaje: ordena (a través de método de inserción) el arreglo por puntaje.
* Insertar: inserta dato en sub-arreglo ordenado.
* Mostrar lista jugadores: muestra la lista de jugadores en el orden seleccionado.

**Problemas Encontrados**

Uno de los problemas encontrados fue la recursividad accidental que se generaba al momento de manejarse entre los distintos menús.

Esto generaba un conflicto al intentar cerrar la sesión y finalizar el programa.

**Solución**

Para solucionarlos, tuvimos que eliminar las llamadas relacionadas a los menús, en el código, evitando la recursividad.

Y para lograr cerrar sesión desde el menú de opciones, sin pasar por el Menú Principal, creamos un puntero que mande directo a cerrar sesión.