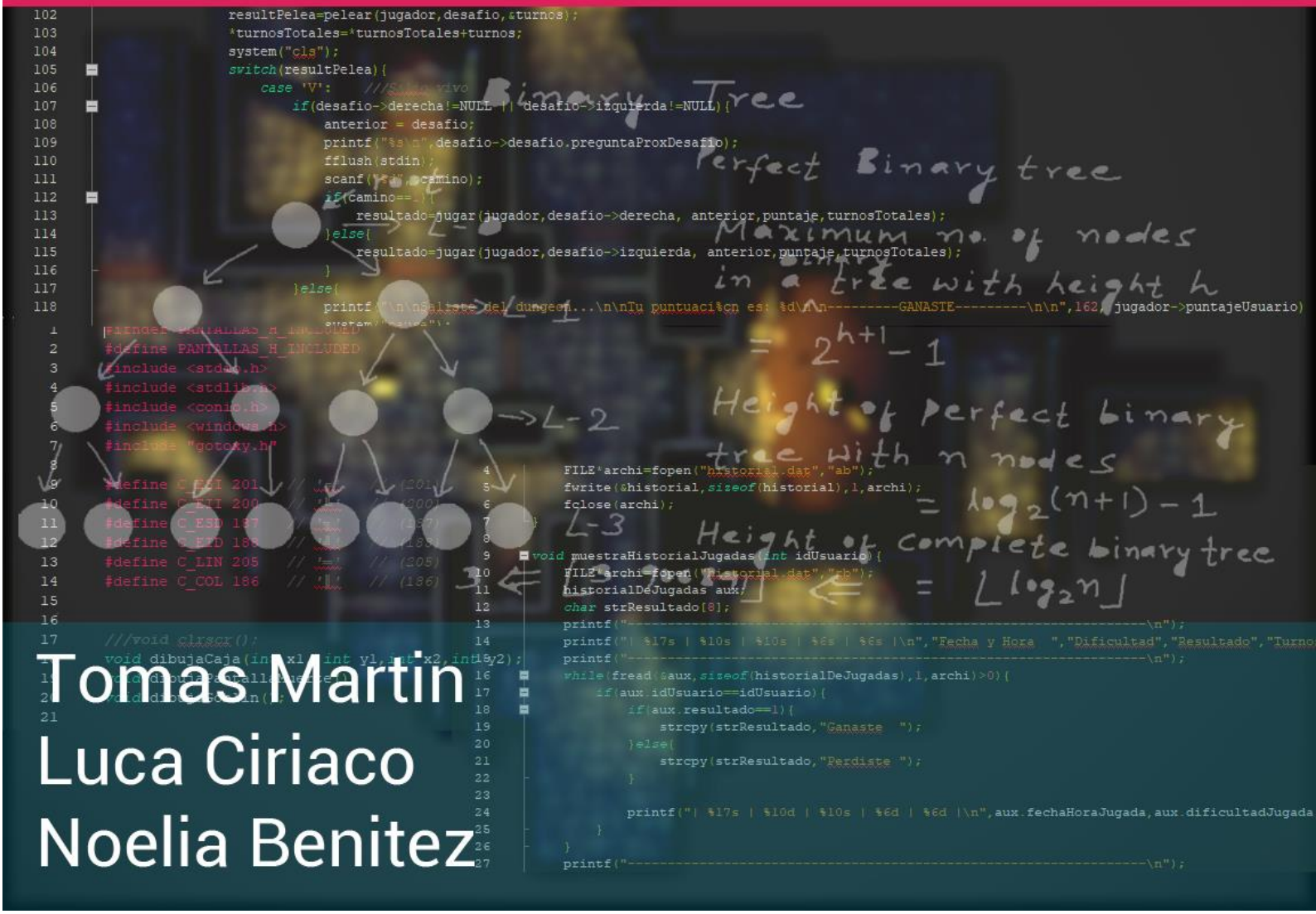
Tree-Dungeon-Crawler

Trabajo practico final programación 2



Tomas Martin

Luca Ciriaco

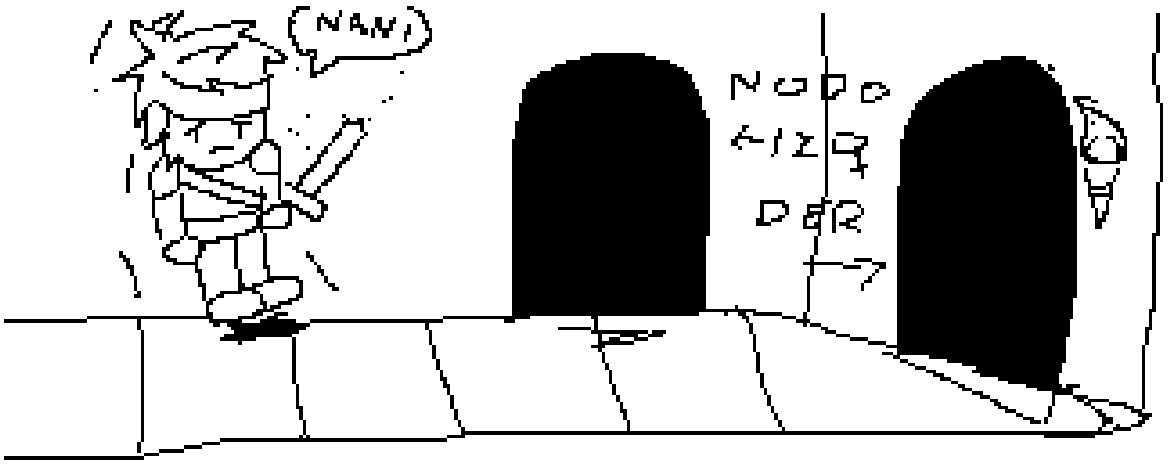
Noelia Benitez

Glosario

De qué trata el trabajo practico:

El trabajo práctico consiste en hacer un juego por turnos donde el jugador tiene que recorrer un dungeon que está conformado por un árbol binario. En cada nodo del árbol binario (encuentro) tendrá enfrentamientos con monstruos o recompensas (sacados de una lista dinámica). Luego de terminar la sesión de juego el jugador podrá comparar sus puntajes en un ranking y ver sus estadísticas.

Por otro lado también planteamos una función de administrador para que se puedan agregar editar eliminar monstruos dungeons y usuarios.



Manual de usuario:

Registrarse:

Para registrarte ir a la opción [1] del menú principal, luego de eso ingrese su nombre de usuario y contraseña.

Iniciar sesión:

Para iniciar sesión ir a la opción [2] del menú principal, luego de eso ingrese su nombre de usuario y contraseña.

Jugar:

Para poder jugar primero debes iniciar sesión en el menú principal, a continuación de ingresar su nombre y contraseña se ingresa al submenú de jugar y ahí elija la opción [1] de jugar.

Como jugar:

El jugador empieza con una breve narración de ahí parte a la toma de decisiones, como la opción de desplazarse por el dungeon (árbol binario) de ahí tendrá encuentros con monstruos.

En cada encuentro con un monstruo el jugador tiene dos opciones

1) Atacar al monstruo:

El jugador tirara un numero aleatorio de daño para hacerle al monstruo y si el daño excede a la vida del monstruo el monstruo muere y el jugador será recompensado con una suma puntos.

2) Huir:

El jugador huye del enfrentamiento con el monstruo y se dirige al nodo anterior. Si el jugador elige esta opción será penado con una resta de puntos.

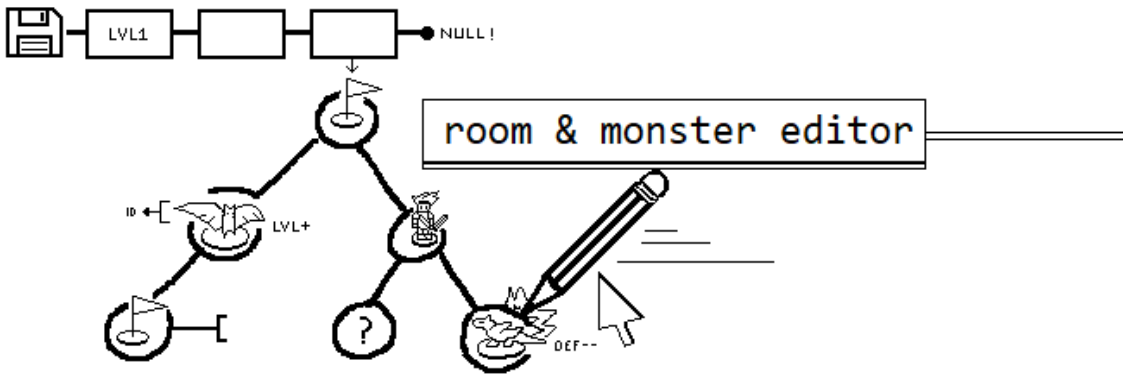
Ver historial de jugadas:

Para poder ver el historial de jugadas debes iniciar sesión en el menú principal, a continuación de ingresar su nombre y contraseña se ingresa al submenú de jugar y ahí elija la opción [3] de ver el historial de jugadas.

Ver mejores puntajes:

Para poder los mejores puntajes debes iniciar sesión en el menú principal, a continuación de ingresar su nombre y contraseña se ingresa al submenú de jugar y ahí elija la opción [2] de ver mejores puntajes.

Manual de Administrador:



Agregar/editar/modificar/mostrar desafíos:

Para Agregar/eliminar/modificar/mostrar un desafío primero debes iniciar sesión como modo administrador en el menú principal, una vez iniciado sesión se ingresa al submenú de jugar y ahí al submenú de ABML desafío.

Una vez en el menú ABML desafío podrá elegir las opciones de:

[1] Alta

Aclaración para agregar un monstruo al desafío primero debe estar dado de alta

[2] Baja

[3] Modificación

[4] Listado

[5] Volver al menú anterior

Agregar/editar/modificar/mostrar monstruos:

Para Agregar/eliminar/modificar/mostrar un monstruo primero debes iniciar sesión como modo administrador en el menú principal, una vez iniciado sesión se ingresa al submenú de jugar y ahí al submenú de ABML monstruo.

Una vez en el menú ABML monstruo podrá elegir las opciones de:

[1] Alta

[2] Baja

[3] Modificación

[4] Listado

[5] Volver al menú anterior

Diario de trabajo

	Descripción
Día	Diseño de estructuras.
25-10	Creación de board en Trello.
	Creación de repositorio en GitHub.
Día	Funciones básicas de usuario.
29-10	Funciones básicas de desafío.
	Funciones de pelear/jugar.
	Funciones de historial de jugadas.
	Funciones de menú.
Día	Cambios en funciones de desafíos.
31-10	
Día	Funciones de listas de monstruos.
1-11	Ajustes de diferentes funciones.
Día	Más ajustes de diferentes funciones.
4-11	
Día	Funciones de usuarios con archivos.
6-11	
Día	Más ajustes de diferentes funciones.
7-11	
Día	Función de huida.
8-11	Más arreglos de diferentes funciones.
	Arreglos visuales para la consola.
Día	Lista de monstruos dentro del árbol desafió.
9-11	
Día	Historial de jugadas.
16-11	Arreglos gráficos.
	Librería de puntajes.
Día	ABML Monstruos.
17-11	ABML Desafíos.

	Descripción
Día	
17-11	Arreglo librería de puntajes.
	Carga de desafíos desde archivo.
	Carga de dibujos de monstruos desde archivos de texto.
	Últimos retoques a la documentación.
	Pulido general del programa.
Día	Entrega y exposición del trabajo practico
18-11	

Detalles de estructuras y funcionalidades básicas:

Estructura 1

```
typedef struct{
   int idDesafio;
   char tipoDesafio;
   char descripcionDesafio[100];
   int dificultadDesafio;
   STmonstruo monstruo;
   char preguntaProxDesafio[100];
   int desafioEliminado;
}STdesafio;

typedef struct{
STdesafio desafio;
struct nodoArbolDesa*izquierda;
struct nodoArbolDesa*derecha;
}nodoArbolDesa;
```

Funciones asociadas:

Utilizamos esta estructura como parte del dungeon donde el jugador lo recorre un árbol binario.
También está compuesta por un arreglo de árboles donde cada árbol del arreglo representa una dificultad diferente.

```
nodoArbolDesa * inicArbolDesafio();
nodoArbolDesa * crearNodoArbolDesafio(STdesafio desafio);
nodoArbolDesa * insertarNodoArbolDesafio(nodoArbolDesa*arbol,STdesafio desafio);
void mostrarNodoArbolDesafio(nodoArbolDesa*arbol);
void listarArbolDesafio(nodoArbolDesa*arbol);
void guardarDesafioEnArchivo (nodoArbolDesa* nuevo);
nodoArbolDesa* pasarDesafiosArbolToArbol(nodoArbolDesa* arbol);
```

Estructura 2

```
typedef struct{
   int idMonstruo;
   char nombreMonstruo[30];
   int vidaBaseMonstruo;
   int ataqueBaseMonstruo;
   int puntosMonstruo;
}STmonstruo;

typedef struct{
STmonstruo monstruo;
struct nodoMonstruo* sig;
struct nodoMostruo* ante;
}nodoMonstruo;
```

Funciones asociadas:

Utilizamos esta estructura para guardar las características de cada monstruo y después pasarlas al árbol binario para poder enfrentar al jugador

```
void agregarMonstruoArchivo (STmonstruo aux);
nodoMonstruo * cargarListaMonstruos(nodoMonstruo* lista);
nodoMonstruo * crearNodoMonstruo(STmonstruo monstruo);
void mostrarMonstruo(STmonstruo aux);
void recorrerMostrar(nodoMonstruo * listaMonstruos);
nodoMonstruo* agregarFinal(nodoMonstruo* listaMonstruos, nodoMonstruo * nuevoMonstruo);
nodoMonstruo * buscarUltimo(nodoMonstruo * listaMonstruos);
nodoMonstruo* buscarMonstruoNombre(nodoMonstruo* lista, char nombre[]);
STmonstruo ponerMonstruo(nodoMonstruo* lista, char nombre[], int dificultad);
```

Estructura 3

```
typedef struct{
char nombre[30];
int puntaje;
}STpunt0ajes;

typedef struct{
STpuntajes puntajes;
struct nodoPuntajes * siguiente;
}nodoPuntajes;
```

Funciones asociadas:

Utilizamos esta estructura para almacenar los puntajes de cada partida jugada

```
nodoPuntajes * inicListaPuntajes();
nodoPuntajes * crearNodoPuntajes(int puntaje,char nombre[30]);
nodoPuntajes * acomodarPuntaje(nodoPuntajes * lista, nodoPuntajes * nuevo);
nodoPuntajes * abrirArchivoPuntajes();
void guardarArchivoPuntajes(nodoPuntajes * lista);
void cargarPuntajes(char nombre[30],int puntaje);
```

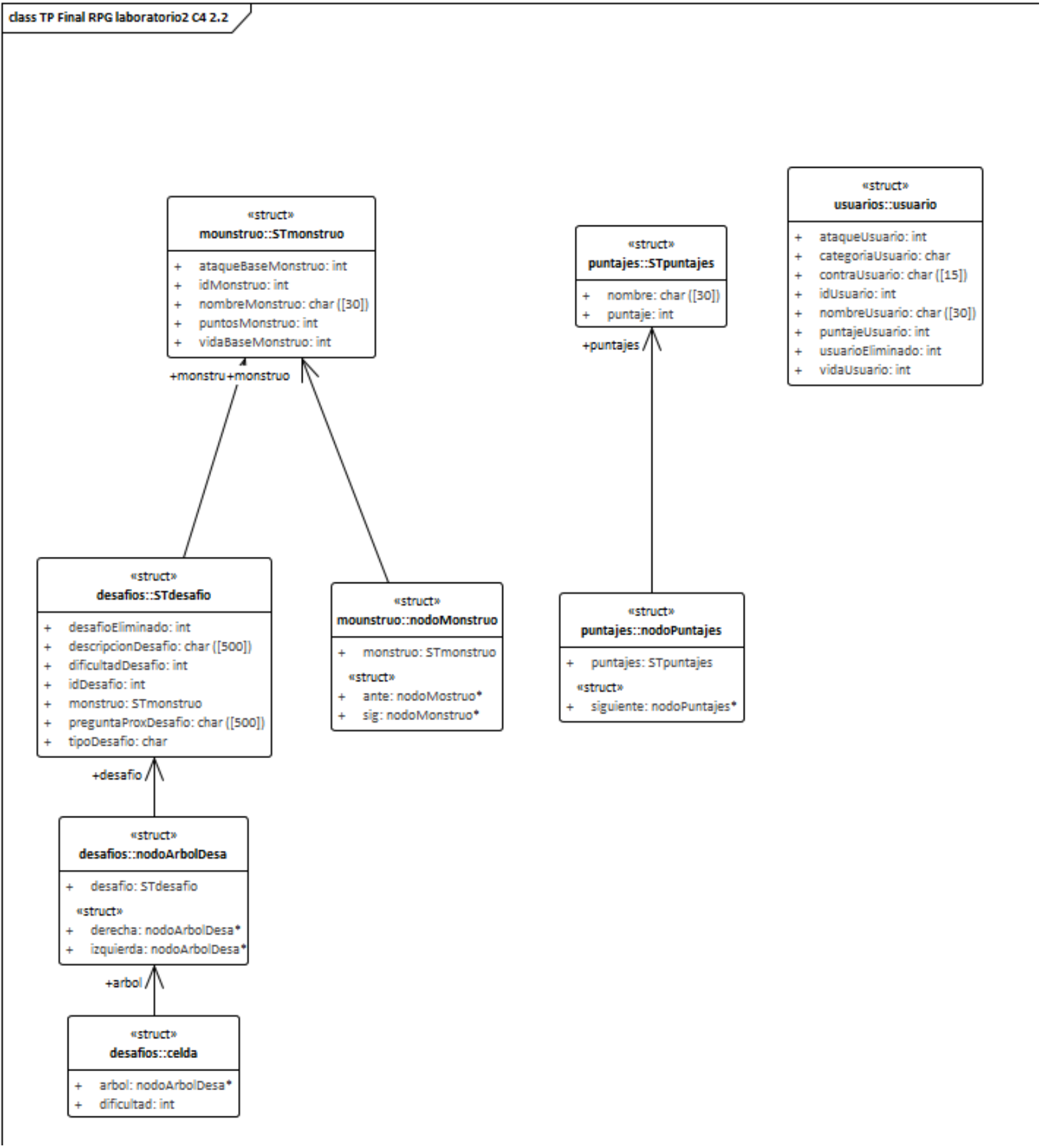
Estructura 4

```
typedef struct{
   int idUsuario;
   char fechaHoraJugada[17];
   int dificultadJugada;
   int resultado;
   int turnosTotalesJugados;
   int puntosGanados;
}historialDeJugadas;
```

Funciones asociadas:

Utilizamos esta estructura para sacar estadísticas de cada partida jugada

```
void muestraHistorialJugadas(int idUsuario);
void guardarHitoricoJugadas(historialDeJugadas historial);
```


class TP Final RPG laboratorio2 C4 2.2
«struct»
mounstruo::STmonstruo
+ ataqueBaseMonstruo: int
+ idMonstruo: int
+ nombreMonstruo: char ([30])
+ puntosMonstruo: int
+ vidaBaseMonstruo: int
«struct»
puntajes::STpuntajes
+ nombre: char ([30])
+ puntaje: int
«struct»
usuarios::usuario
+ ataqueUsuario: int
+ categoriaUsuario: char
+ contraUsuario: char ([15])
+ idUsuario: int
+ nombreUsuario: char ([30])
+ puntajeUsuario: int
+ usuarioEliminado: int
+ vidaUsuario: int
+monstru
+monstruo
+puntajes
«struct»
desafios::STdesafio
+ desafioEliminado: int
+ descripcionDesafio: char ([500])
+ dificultadDesafio: int
+ idDesafio: int
+ monstruo: STmonstruo
+ preguntaProxDesafio: char ([500])
+ tipoDesafio: char
«struct»
mounstruo::nodoMonstruo
+ monstruo: STmonstruo
«struct»
+ ante: nodoMostruo*
+ sig: nodoMonstruo*
«struct»
puntajes::nodoPuntajes
+ puntajes: STpuntajes
«struct»
+ siguiente: nodoPuntajes*
+desafio
«struct»
desafios::nodoArbolDesa
+ desafio: STdesafio
«struct»
+ derecha: nodoArbolDesa*
+ izquierda: nodoArbolDesa*
+arbol
«struct»
desafios::celda
+ arbol: nodoArbolDesa*
+ dificultad: int

class TP Final RPG laboratorio2 C4 2.2

mounstruo

+ agregarFinal(nodoMonstruo*, nodoMonstruo*): nodoMonstruo*
+ agregarMonstruoArchivo(STmonstruo): void
+ bajaMonstruo(nodoMonstruo*, char): nodoMonstruo*
+ buscarMonstruoNombre(nodoMonstruo*, char): nodoMonstruo*
+ buscarMonstruoPorId(int, STmonstruo*): void
+ buscarUltimo(nodoMonstruo*): nodoMonstruo *
+ cantMonstruosEnArchivo(): int
+ cargarListaMonstruos(nodoMonstruo*): nodoMonstruo *
+ cargarMonstruo(): STmonstruo
+ crearNodoMonstruo(STmonstruo): nodoMonstruo *
+ iniclista(): nodoMonstruo *
+ modificarAtaqueMonstruo(nodoMonstruo*, nodoMonstruo*): void
+ modificarPuntosMonstruo(nodoMonstruo*, nodoMonstruo*): void
+ modificarRegistroMonstruo(char, int, int): void
+ modificarVidaMonstruo(nodoMonstruo*, nodoMonstruo*): void
+ monstruoVacio(): STmonstruo
+ mostrarListaMonstruo(nodoMonstruo*): void
+ mostrarMonstruo(STmonstruo): void
+ pasarArchivoMonstruosToLista(nodoMonstruo*): nodoMonstruo*
+ pasarListaMonstruoToArchivo(nodoMonstruo*): void
+ ponerMonstruo(nodoMonstruo*, int): STmonstruo

desafios

+ altaREGdesafio(): void
+ buscarDesafioEnArbolID(nodoArbolDesa*, int): nodoArbolDesa*
- buscarYReemplazarREGDesafio(int, REGdesafio): void
+ crearNodoArbolDesafio(STdesafio): nodoArbolDesa*
- existeREGDesafioID(int): int
+ guardarDesafioEnArchivo(nodoArbolDesa*): void
+ inicArbolDesafio(): nodoArbolDesa*
+ insertarNodoArbolDesafio(nodoArbolDesa*, STdesafio): nodoArbolDesa*
+ listarArbolDesafio(nodoArbolDesa*): void
+ mostrarNodoArbolDesafio(STdesafio): void
+ muestraArchiDesafios(): void
- muestraArchiMonstruos(): void
+ pasarDesafiosArbolToArbol(nodoArbolDesa*): nodoArbolDesa*
- pasarDesafiosArchivoToArbol(celda): void
+ reemplazarNodoDesafio(nodoArbolDesa*, REGdesafio): void
+ REGparaReemplazar(int): REGdesafio

«struct» desafios::REGdesafio

+ desafioEliminado: int
+ descripcionDesafio: char ([500])
+ dificultadDesafio: int
+ idDesafio: int
+ idMonstruo: int
+ preguntaProxDesafio: char ([500])
+ tipoDesafio: char

gotoxy

+ color(int): void
+ gotoxy(int, int): void
+ hidecursor(int): void
+ whereX(): int
+ whereY(): int

usuarios

+ bajaUsuario(char): void
+ cantUsuariosEnArchivo(): int
+ convertirJugadorToAdmin(char): void
+ guardarNuevoUsuArchivo(usuario): void
+ mostrarArchivoUsu(): void
+ mostrarUsuario(usuario): void
+ posUsuarioNombreEnArchivo(char): int
+ usuarioPorRegistro(int): usuario

Pantallas

+ dibujaCaja(int, int, int, int): void
+ dibujaGoblin(): void
- dibujaMonstruoDesdeTXT(int, int, char): void
+ dibujaPantallaMuerte(): void

menu

+ ABMLdesafios(): void
+ ABMLmonstruos(nodoMonstruo*): void
+ administrarUsuarios(): void
+ buscarYReemplazarREGDesafio(int, REGdesafio): void
+ existeREGDesafioID(int): int
+ getfechayhora(char): void
+ iniciarPrograma(): void
+ logInUser(celda, nodoMonstruo*): void
+ menuPrincipal(celda, nodoMonstruo*): void
+ menuUsuario(usuario, celda, nodoMonstruo*): void
+ modificarMonstruo(nodoMonstruo*, nodoMonstruo*): void
+ nuevoUsuario(celda, nodoMonstruo*): void
+ pantallaPrincipal(): void
+ REGparaReemplazar(int): REGdesafio

«struct» historialDeJugadas:: historialDeJugadas

+ dificultadJugada: int
+ fechaHoraJugada: char ([17])
+ idUsuario: int
+ puntosGanados: int
+ resultado: int
+ turnosTotalesJugados: int

jugar

+ calculoDanio(int): int
+ jugar(usuario*, nodoArbolDesa*, nodoArbolDesa*, int*, int*): int
+ pelear(usuario*, nodoArbolDesa*, int*): char
- recompensa(usuario*): void

puntajes

+ abrirArchivoPuntajes(): nodoPuntajes *
+ acomodarPuntaje(nodoPuntajes*, nodoPuntajes*): nodoPuntajes *
+ agregarPpio(nodoPuntajes*, nodoPuntajes*): nodoPuntajes*
+ cargarPuntajes(char, int): void
+ crearNodoPuntajes(char, int): nodoPuntajes *
+ guardarArchivoPuntajes(nodoPuntajes*): void
+ inicListaPuntajes(): nodoPuntajes *
+ mostrarListaPuntajes(nodoPuntajes*): void
+ mostrarPuntajes(STpuntajes, int): void
+ recorreMostrarPuntajes(): void

historialDeJugadas

+ guardarHitoricoJugadas(historialDeJugadas): void
+ muestraHistorialJugadas(int): void

Matriz de soluciones:

Problema	Solución
Descargábamos todo el archivo de usuarios a una lista dinámica y fue ineficiente al momento de manejarlo.	Solo descargamos el usuario que ingresa a la cuenta.
La consola mostraba erróneamente los gráficos del juego debido a las resoluciones de pantalla en los que fueron probados.	Hacer una función que el tamaño de la consola.
Al modificar algo no comiteado en el repositorio de GitHub y tratar de pullear una nueva versión con cambios en la misma línea de código se generaba un error de conflicto entre versiones.	Al abrir GitHub se tenía que elegir el cambio que se quería que se guarde en el master.
Uno de los monstruos seguía viviendo con vida negativa.	Despues de terminar la sesión de juego se vuelve a cargar el árbol desde el archivo así se vuelven a setear los puntos de vida.
En el nodo raíz del árbol se podía huir (ir para nodo anterior).	No puede haber una pelea en el primer nodo por que no se puede ir para atrás del nodo raíz del árbol.
Al momento de cargar el archivo de puntajes el programa se colgaba y se interrumpía el proceso.	Los parámetros de la función de guardar archivo estaban puestos al revés.

Herramientas utilizadas y sus funciones.

Codeblocks:

Editor de código.

VS Code:

Editor de código cuando había conflictos de versiones en el repo.

GitHub Desktop:

Repositorio y control de versión.

Trello:

Herramienta de metodología ágil kanban.

Discord:

Conferencias grupales.

Word:

Documentación.

Enterprise Architect:

Diagrama de estructuras y funciones.

Notepad++:

Editor de texto para ver imágenes hechas en ASCII.