Proyecto Integrador: Algoritmos y Estructuras de Datos II

Profesoras: Ana María Company
Jaqueline Escalante
Yamila Aquino

Alumnos: Gonzalez Ovideo Bruno Fabricio
Rodas Mario Daniel
Grupo 17

Objetivos del proyecto:

Objetivo Desarrollar un proyecto de software en C que integre estructuras de datos dinámicas, técnicas de programación avanzadas y TADs, aplicando los conceptos de la materia a un problema elegido por el grupo de no más de 4 integrantes (con creatividad).

Herramientas Técnicas utilizadas:

- Tecnicas de programacion.
- Recursividad.
- TADs.
- Manipulacion de archivos.

Tema y contexto:

Nuestro programa “Slay the Algorithm” es un juego lineal basado en un sistema de preguntas, vidas, ventajas y puntuación, en el que nos inspiramos en juegos de estilo *roguelite* (estilo de juego en el que, en cada nueva partida, los enemigos son distintos).

Objetivo:

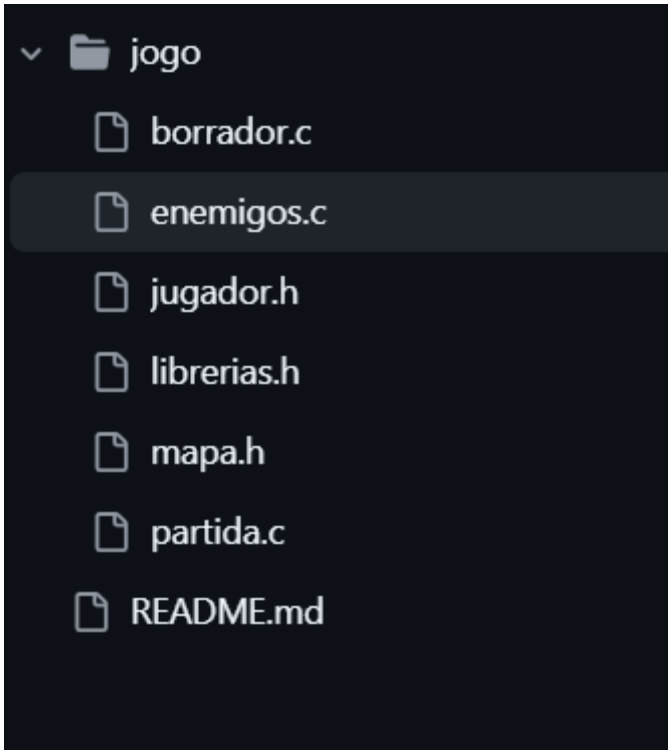
En este caso nos planteamos desarrollar un sistema en el que, utilizando ciertos métodos, cada pregunta pueda resultar distinta o no a la anterior. Nos lo planteamos desarrollando ciertas ideas lógicas mediante el uso y conocimiento que adquirimos a lo largo del año. También utilizamos técnicas para adaptar un sistema de vida y puntuación ligado al sistema de preguntas generado.

Informe:

Consideramos distintos tipos de diseño, ya que al hacerlo a la par pudimos ponernos de acuerdo rápidamente. Es una estructura que iniciamos por nuestra cuenta, pensándola de una manera lógica muy general, utilizando distintos borradores y plasmando ciertas ideas que se iban ocurriendo o que podíamos agregar y mejorar en lo que al código respecta.

```
/*  *Borrador*

    a modo slay the spire juego progresivo y roguelite

     Secciones claves
                * Jugador (menu o interfaz para navegacion)
                * Partida( una seed de generacion aleatoria)
                * Niveles (arboles binarios) y progreso (estructuras condicionales)
                * Boss (algoritmos con diferentes preguntas y tipos )
                * guardado de partidas ( archivos binarios junto con punteros )

                (podrian hacerse en .h para hacerlo mas sencillo de config)

- 1 solo jugador o partida que se vaya guardando
- X niveles con distintos tipos de preguntas
- Dificultad de las preguntas que sea aleatoria (1 a 3 de dificultad y 4 exclusivamente para el Boss)
- Combate de multiple choice
- En una pelea podrian haber maximo 2 enemigos (max 2 de dificultad si son dos enemigos)
- OPCIONAL: diferentes ventajas como:
                                    * cambiar pregunta
                                    * eliminar 2 opciones
                                    *

```

A lo largo del proyecto hubo ciertas cuestiones que, tanto por tiempo como por complejidad, decidimos ir descartando.
La estructura que ideamos está basada en distintas bibliotecas entrelazadas entre sí, en la cual se nos presentó un problema que no prevenimos: el hecho de que una biblioteca estuviera dentro de otra y después de sí misma, generando una llamada recíproca entre las bibliotecas que producía un error y un bucle. Esto se solucionó estructurando todo de una manera más ordenada y organizada, en estilo y forma de "pila", en la cual cada una va llamando a la siguiente hasta llegar a la última. Decidimos armar distintas bibliotecas para organizar de mejor manera la estructura de dónde van o dónde se ubican cada función, haciendo más estructurado el proceso a la hora de buscar ciertas cosas o saber dónde ubicarlas.



Primer protoripo de biblioteca

Consideramos pocas alternativas, ya que hacerlo mediante bibliotecas nos pareció lo más lógico y más seguro para realizar; el hecho de colocar casi todo el código en un solo lugar solo generaría un código spaghetti.

Para ciertas partes del código decidimos utilizar estructuras de datos compuestas para los archivos y el registro del jugador.

```
typedef struct{

        tVidas vida;
        int puntaje;
        int nivel;
        bool ventajaCambio;
        bool ventajaEscudo;
        int enemigosMatados;

}tRegJugador;
```

```
typedef struct{
    tString pregunta;
}tRegArchivo;
```

También utilizamos estructuras de repeticiones iterativas para comprobar que no errara la pregunta o por si el jugador se quedaba sin vidas.

```
    }while(respuestaIncorrecta && acumuladorVidas(*jugador1)>0);
```

Para errores de entrada:

```
            printf("\nElegista una letra equivocada, ingrese denuevo\n");
        }
    }while(!esABCD(respuesta));
```

Nos pareció correcta esta estructura para que, cuando errara la respuesta o errara las opciones de letra que se le piden, vuelva a pedirle que ingrese; pues, en caso de no comprobar esto, el juego se rompería lógicamente.

Además, utilizamos la estructura switch en varios casos, ya que empleamos por lo menos dos opciones y, además, utilizamos la opción default, por lo tanto ya nos cubría por lo menos tres posibles casos, siendo así más eficiente que el if.

```
384             do{
385                 printf("\nPosee Ventaja escudo \n1-Usar\n2-No usar\n");
386                 fflush(stdin);
387                 scanf("%d", &opcion);
388                 switch(opcion){
389                     case 1:{
390                         printf("\nEscudo activado\n");
391                         escudo=true;
392                         actualizarVentaja(jugador1, 2, 2);
393                         break;
394                     }
395                     case 2:{
396                         printf("\nNo se utilizara nada\n");
397                         break;
398                     }
399                     default:printf("\nError: eligio un numero incorrecto\n");
400                 }
401             }while(opcion!=1 && opcion!=2);
```

Además, también usamos el switch a la hora de indicar al programa qué hacer en cada nivel, ya sea gráficamente o en cuestiones de peleas. En este caso nos parece la única opción, ya que hay más de ocho casos en cada uno.

```
103     switch (pNivel){
104         case 1:{
105             nivelDificultad(2, pVida, pPuntaje);
106             *pNivelPregunta=2;
107             break;
108         }
109         case 2:{
110             nivelDificultad(2, pVida, pPuntaje);
111             *pNivelPregunta=2;
112             break;
113         }
114         case 4: {
115             nivelDificultad(1, pVida, pPuntaje);
116             *pNivelPregunta=1;
117             break;
118         }
119         case 6: {
120             nivelDificultad(1, pVida, pPuntaje);
121             *pNivelPregunta=1;
122             break;
123         }
124         case 7:{
```

Le pedimos a Claude.ai, que es una IA más centrada en la programación, que nos genere archivos de tipo .dat en los que se encontrarían las preguntas con las posibles respuestas y un .txt en el que se encontraría la elección de la respuesta correcta. Junto con esto, tuvimos la idea de utilizar vectores para guardarlas y después poder manipularlas. También utilizamos ciertas funciones de las bibliotecas de Dev-C++, tales como system("cls") o exit(EXIT_SUCCESS), para que el programa, a la hora de ejecutarse, resulte agradable y llevadero para la vista.

A lo largo del proyecto no realizamos muchos cambios, ya que al hacerlo a la par llegábamos a un acuerdo en el momento y lo ejecutábamos. Pero hubo tres grandes cambios: uno de ellos fue que originalmente las dificultades de las preguntas iban a ser aleatorias, lo cual luego cambiamos por una elección aleatoria pero dentro de la dificultad.

El segundo cambio fue que las estructuras de los archivos inicialmente iban a tener un índice y iban a estar todas las preguntas de las tres dificultades en el mismo archivo; pero, por recomendación de la Lic. Company, optamos por utilizar un .dat para cada dificultad y un .txt para sus respectivas respuestas, los cuales están en vectores para facilitar su manipulación.

Y por último, el tercer cambio fue que antes los enemigos iban a tener una estructura de datos con vida y su nivel de dificultad; luego cambiamos a que solo tengan una vida y que el nivel de dificultad esté definido por la etapa o nivel del jugador. Esto generó un gran cambio en el sistema de pelea que desarrollamos.

En secciones como “jugador.h”, se utilizaron estructuras de datos compuestos para llevar el registro de todas las estadísticas del jugador. Se iba a utilizar una función que reste y otra que sume, pero llegamos a la conclusión de un resultado lógico que nos permitiera combinar ambas en una a través del pasaje de parámetros, la cual llamamos actualizarVida, y luego utilizamos el mismo método en actualizarPuntaje y actualizarVentaja.

```
42  void actualizarVida(tRegJugador* jugador1, int pActualizacion){
43      int i;
44      if(pActualizacion==2){
45          for(i=0; i<VIDAS; i++){
46              if(jugador1->vida[i]){
47                  jugador1->vida[i]=false;
48                  return;
49              }
50          }
51      }
52      else{
53          for(i=0; i<VIDAS; i++){
54              if(!jugador1->vida[i]){
55                  jugador1->vida[i]=true;
56                  return;
57              }
58          }
59      }
60  }
```

En la biblioteca “enemigos.h”, como habíamos dicho antes, guardamos las preguntas y respuestas en distintos archivos para que sea de fácil acceso y poder manipularlos en el orden que queramos. Se guardan en vectores globales, los cuales utilizan índices para acceder a la pregunta; de esta manera, empleamos métodos lógicos para poder designarlos. Por ejemplo: el siguiente del índice se utilizaba para acceder a las posibles opciones y la mitad del índice para las respuestas.

Se creó un tipo de dato registro que solo contenía tString[200] con el único fin de poder manipularlo como registro a la hora de guardarlo en los archivos y a la hora de castearlos.

Decidimos dividir las preguntas en tres niveles de dificultad para generar una diversidad con respecto a los enemigos (preguntas) y así generar diferentes casos o escenarios en la jugabilidad.

```
17  //Tipos de datos para vectores con preguntas y respuestas
18  typedef tString tVectorPreguntas[TAMANIOVECTORPREGUNTAS];
19  typedef char tVectorRespuestas[TAMANIOVECTORRESPUESTAS];
20
21  //variables globales de vectores con preguntas y respuestas
22  tVectorPreguntas vectorPreguntasFaciles, vectorPreguntasNormales, vectorPreguntasDificiles;
23  tVectorRespuestas vectorRespuestasFaciles, vectorRespuestasNormales, vectorRespuestasDificiles;
24
```

En el apartado “mapa.h” (más específicamente en la función ‘tienda’) tuvimos que realizar distintas comparaciones con respecto a las ventajas para poder abarcar todos los escenarios posibles: en caso de que poseía una de las dos, cuál es la que tiene, y en caso de que no poseía ninguna.

Como último paréntesis, a la hora final de poner en revisión todo el progreso armado antes de la entrega, recurrimos a la IA como herramienta para que nos aconsejara qué partes podríamos optimizar o mejorar de nuestro código. Esta nos aconsejó ciertas partes con respecto a la recursión, la cual implementamos, y se nos ocurrió la idea de agregar un “ranking.h” basándonos en lo que la IA nos fue asesorando para poder lograrlo, utilizando un método de árboles, búsqueda y ordenamiento, el cual nos resultó efectivo y completamente funcional para nuestro código.

Funcionalidades implementadas:

- Sistema de Preguntas: 20 preguntas por dificultad elegidas aleatoriamente.
- Sistema de combate: Respuestas correctas o incorrectas.
- Gestión de vida: 5 vidas iniciales con pérdida de 1 por cada error.
- Sistema de puntaje: Suma por acierto y resta por ventaja.
- Progresión: 9 niveles.
- Mapa visual en ASCII: Estados diferentes del mapa.
- Campamento: Restaura 1 vida.
- Tienda: Compra ventajas con puntos.
- Enemigos de Elite: 2 preguntas consecutivas.
- Boss final: 3 preguntas difíciles.
- Ventajas: Escudo/Cambio.
- Ranking: TOP 10 con persistencia.

CONCLUSIONES

Fabricio:

En lo personal considero que el proyecto me ayudó a afianzar los conocimientos que ya tenia, ya que al usarlos de manera mas compleja y con un objetivo, pude interiorizar esos conocimientos.

Además trabajar con mi compañero fue muy importante ya que entre los dos dabamos ideas, tomabamos decisiones, nos ayudabamos con problemas, y todo eso hizo que sea mas simple el trabajo y mas llevadero. Además de que los disfrutamos haciendo y mas cuando salian bien las cosas.

Por otro lado el conocimiento que mas aprendimos fue el de las librerias, pues al separar todo el proyecto en librerias fue todo un desafio lograr que se conecten sin errores, con varios puntos de aprendizaje en el camino tales como evitar bucles o conexiones con tipos de datos entre librerias.

Por último lo que me gustó o lo que mas me sirvio es que, esta materia da pie para muchas otras cosas en el futuro, los temas en general utilizados son muy utiles, pero por sobre todo los archivos externos me parecen un paso muy importante, pues la persistencia de los datos es algo que yo considero super necesario para poder avanzar en mis conocimientos y poder formarme profesionalmente.

Mario:

El desarrollo del juego fue tanto estresante como gratificante ya que también nos permitió ejercitar la creatividad. Transformar un simple sistema de preguntas en un juego tipo roguelite y hacerlo funcional fue todo un desafio. A la vez, integrar conceptos como vectores globales, estructuras compuestas, validaciones de entrada, persistencia de datos y finalmente arboles puso a prueba todos nuestros conocimientos adquiridos

Otro apartado fundamental fue el trabajo en equipo. El hecho de tomar decisiones en conjunto, debatir ideas y dividir responsabilidades marcó una diferencia positiva en la dinámica del proyecto. Esto no solo alivianó la carga individual, sino que también aportó perspectivas distintas que influenciaron en el producto final. En ese sentido, la experiencia refleja con fidelidad el tipo de colaboración que suele darse en proyectos reales de software.

Este proyecto funcionó como un puente entre lo aprendido en clase y su aplicación práctica. No solo reafirmó conceptos teóricos, sino que mostró la importancia de la organización, la documentación, la modularidad y el pensamiento lógico.

En síntesis, el proyecto no solo fortaleció conocimientos técnicos, sino que también incentivo la responsabilidad, la creatividad y el criterio profesional. Fue un desafío que exigió paciencia y análisis. El resultado demuestra un avance significativo tanto en lo académico como en lo personal.