Trabajo Práctico Especial
Programación Imperativa
Primer Cuatrimestre 2010

1. Objetivo.

Diseñar e implementar un juego de rol o *role playing game* (RPG) en modo texto.

2. Descripción del juego.

Los juegos de rol son una clase de juego donde el jugador toma el control de un personaje (también llamado *Avatar*) en un mundo ficticio y realiza acciones sujetas a un sistema de reglas. En el presente Trabajo Práctico Especial, se implementará un sub-género de los juegos de rol conocido como *dungeon crawler* donde los jugadores asumen el "rol" de un héroe inmerso en un laberinto o *dungeon* habitado por monstruos, el cual deben recorrer hasta encontrar la salida.

Los datos con la información del juego se encuentran definidos dentro de un archivo de texto con la siguiente estructura (*los espacios al principio de cada línea se agregaron por claridad, son opcionales y en caso de figurar en el archivo deben ser ignorados al procesarlo. Se asume también a lo largo de este documento que los textos que -por una cuestión de espacio- ocupan más de un renglón, se encuentran en el archivo de texto dentro de una única línea)*:

```
<Juego>
    <PuntosImportantes>
            <HabitacionInicioID>0</HabitacionInicioID>
            <HabitacionSalidaID>3</HabitacionSalidaID>
    </PuntosImportantes>

    <Profesiones>
            <Cantidad>3</Cantidad>
            <Profesion>
                    <ID>0</ID>
                    <Nombre>Guerrero</Nombre>
                    <MinHP>10</MinHP>
                    <MaxHP>30</MaxHP>
                    <MinDP>2</MinDP>
                    <MaxDP>6</MaxDP>
            </Profesion>
            <Profesion>
                    <ID>1</ID>
                    <Nombre>Mago</Nombre>
                    <MinHP>10</MinHP>
                    <MaxHP>30</MaxHP>
                    <MinDP>2</MinDP>
                    <MaxDP>6</MaxDP>
            </Profesion>
            <Profesion>
                    <ID>0</ID>
                    <Nombre>Ladron</Nombre>
                    <MinHP>10</MinHP>
                    <MaxHP>30</MaxHP>
                    <MinDP>2</MinDP>
                    <MaxDP>6</MaxDP>
            </Profesion>
    </Profesiones>

    <Enemigos>
            <Cantidad>2</Cantidad>
```

```
		<Enemigo>
			<ID>0</ID>
			<Nombre>Orco salvaje</Nombre>
			<MinHP>2</MinHP>
			<MaxHP>5</MaxHP>
			<MinDP-0>1</MinDP-0>
			<MaxDP-0>3</MaxDP-0>
			<MinDP-1>2</MinDP-1>
			<MaxDP-1>4</MaxDP-1>
			<MinDP-2>3</MinDP-2>
			<MaxDP-2>6</MaxDP-2>
		</Enemigo>
		<Enemigo>
			<ID>1</ID>
			<Nombre>Orco guerrero</Nombre>
			<MinHP>3</MinHP>
			<MaxHP>7</MaxHP>
			<MinDP-0>2</MinDP-0>
			<MaxDP-0>5</MaxDP-0>
			<MinDP-1>1</MinDP-1>
			<MaxDP-1>3</MaxDP-1>
			<MinDP-2>2</MinDP-2>
			<MaxDP-2>4</MaxDP-2>
		</Enemigo>
	</Enemigos>

	<Laberinto>
		<Cantidad>4</Cantidad>
		<Habitacion>
			<ID>0</ID>
			<Nombre>Entrada</Nombre>
			<Descripcion>Es una habitación relativamente oscura, donde la única iluminación es el ténue brillo de unas pocas velas.</Descripcion>
			<Puertas>
				<Puerta>
					<Nombre>Dorada</Nombre>
					<Destino>1</Destino>
				</Puerta>
				<Puerta>
					<Nombre>Plateada</Nombre>
					<Destino>2</Destino>
				</Puerta>
			</Puertas>
		</Habitacion>
		<Habitacion>
			<ID>1</ID>
			<Nombre>Sala de Armas</Nombre>
			<Descripcion>Texto con la descripción de la sala de armas.</Descripcion>
			<Puertas>
				<Puerta>
					<Nombre>Dorada</Nombre>
					<Destino>0</Destino>
				</Puerta>
				<Puerta>
					<Nombre>Bronce</Nombre>
					<Destino>2</Destino>
				</Puerta>
```

```
                </Puertas>
                <Enemigos>
                        <Cantidad>1</Cantidad>
                        <Enemigo>
                                <ID>0</ID>
                        </Enemigo>
                </Enemigos>
        </Habitacion>
        <Habitacion>
                <ID>2</ID>
                <Nombre>Habitación 2</Nombre>
                <Descripcion>Texto con la descripción de la habitación 2.</Descripcion>
                <Puertas>
                        <Puerta>
                                <Nombre>Plateada</Nombre>
                                <Destino>0</Destino>
                        </Puerta>
                        <Puerta>
                                <Nombre>Bronce</Nombre>
                                <Destino>1</Destino>
                        </Puerta>
                        <Puerta>
                                <Nombre>Platino</Nombre>
                                <Destino>3</Destino>
                        </Puerta>
                </Puertas>
                <Enemigos>
                        <Cantidad>2</Cantidad>
                        <Enemigo>
                                <ID>1</ID>
                        </Enemigo>
                        <Enemigo>
                                <ID>0</ID>
                        </Enemigo>
                </Enemigos>
        </Habitacion>
        <Habitacion>
                <ID>3</ID>
                <Nombre>Salida</Nombre>
                <Descripcion>Descripción de la habitación de salida.</Descripcion>
        </Habitacion>
    </Laberinto>
</Juego>
```

Un juego se define como una serie de puntos importantes, de profesiones, de enemigos y un laberinto.

Los **puntos importantes** son:

- **HabitacionInicioID:** Número entero mayor o igual que cero con el identificador de habitación donde el *Avatar* comienza su aventura.
- **HabitacionSalidaID:** Número entero mayor o igual que cero con el identificador de habitación donde el *Avatar* finaliza su aventura.

Nótese que los IDs deben ser distintos.

Las **profesiones** indican el alineamiento del jugador y determinan su desempeño al ser atacados por enemigos. A la hora del combate, ciertas profesiones tienen ventajas respecto a determinados tipos de enemigos. El grupo de profesiones –cuya cantidad de elementos se define mediante el campo **Cantidad**- tiene al menos una profesión, la cual se define mediante los siguientes componentes:

- **ID:** Un identificador numérico único mayor o igual que cero.
- **Nombre:** Un string que indica el nombre genérico de la profesión (ej: "Guerrero", "Mago", etc).
- **MinHP:** Número entero mayor que cero con la cantidad mínima de puntos de vida con los que cuenta el jugador al comenzar el partido.
- **MaxHP:** Número entero mayor que cero con la cantidad máxima de puntos de vida con los que cuenta el jugador al comenzar el partido.
- **MinDP:** Número entero mayor que cero con la cantidad mínima de puntos de daño que realiza el jugador al efectuar un ataque. Se asume que el jugador cuenta con una única arma.
- **MaxDP:** Número entero mayor que cero con la cantidad máxima de puntos de daño que realiza el jugador al efectuar un ataque. Se asume que el jugador cuenta con una única arma.

El conjunto de **enemigos** –cuya cantidad de elementos se define mediante el campo **Cantidad**- cuenta con la definición de uno o más tipos de enemigos únicos. Un tipo de enemigo se define a partir de los siguientes componentes:

- **ID:** Un identificador numérico único mayor o igual que cero.
- **Nombre:** Un string que identifica el nombre genérico del enemigo (ej: "Ogro guerrero", "Goblin asesino", etc).
- **MinHP, MaxHP:** Ídem a los atributos definidos para el jugador.
- **MinDP-n y MaxDP-n:** Ídem a los atributos definidos para el jugador, con la diferencia de que "n" es un número entero entre **0** y **número de profesiones – 1** que indica los puntos mínimos y máximos de daño que produce dicho enemigo a aquellos *avatares* con esa profesión.

Un **laberinto** es un conjunto de dos o más habitaciones. El número de habitaciones se define mediante el parámetro **Cantidad**, y cada habitación se define con los siguientes componentes:

- **ID:** Un identificador numérico único mayor o igual que cero.
- **Nombre:** Un string con el nombre de la habitación.
- **Descripción:** Un string con la descripción de la habitación.
- **Puertas:** Puede ser una o más (opcional).
- **Enemigos:** Puede ser uno o más (opcional).

Una **puerta** se define mediante los siguientes componentes:

- **Nombre:** Un string con el nombre que la identifica.
- **Destino:** Un valor numérico con la habitación destino a la que conduce.

Un **enemigo** dentro de una habitación del laberinto se define mediante el siguiente componente:

- **ID:** Un número entero que hace referencia a uno de los enemigos previamente definidos.

Mediante este sistema, los tipos de enemigos se definen una única vez dentro del listado Enemigos y luego se utilizan una o más veces en las distintas habitaciones del laberinto, evitando redundancia de datos.

Nótese que el hecho de utilizar archivos de texto proporciona una inmensa flexibilidad a la hora de definir los laberintos. De esta forma, no es necesario “cablear” los laberintos dentro del código fuente del programa, lo cual permite utilizar el mismo ejecutable para cualquier laberinto cambiando sólo el archivo de texto.

3. Descripción funcional del programa.

Al iniciar el programa, éste deberá:

1. Validar que el archivo donde está definido el mapa sea correcto (posee todos los campos, para cada enemigo se define un **MinDP** y un **MaxDP** para cada tipo de profesión del jugador, etc). Informar con un mensaje de error y salir del programa si hubiera alguna irregularidad. A medida que se va leyendo el archivo de definición del mapa, crear una matriz tridimensional donde se almacenen los puntos de daño mínimos y máximos (**MinDP** y **MaxDP**) que ocasionaria cada tipo de enemigo sobre cada tipo de profesión. La primera dimensión de la matriz corresponde al tipo de enemigo, la segunda corresponde al tipo de profesión y la tercera –de tamaño fijo 2- corresponde a los puntos de daño mínimos y máximos. Nótese que la matriz debe ser creada dinámicamente, puesto que se desconoce en tiempo de compilación la cantidad de enemigos y profesiones que van a ser utilizadas a lo largo de la partida.
2. Solicitar al jugador que ingrese el nombre de su *Avatar* (máximo 32 caracteres, que pueden incluir cualquier símbolo del alfabeto Inglés).
3. Mostrar un menú con las distintas profesiones disponibles utilizando los números entre **1** y **número de profesiones**. Solicitar al jugador que elija una profesión para su *Avatar*.
4. Crear de manera pseudo-aleatoria los atributos del *Avatar* utilizando las siguientes fórmulas:
    a. Puntos de vida o *hit points* (**HP**): Número pseudo-aleatorio entre **MinHP** y **MaxHP** (**MinHP** y **MaxHP** dependen de la profesión elegida).
    b. Puntos de daño o *damage points* (**DP**): Número pseudo-aleatorio entre **MinDP** y **MaxDP** (**MinDP** y **MaxDP** dependen de la profesión elegida).
5. Situar al jugador en la habitación cuyo ID sea igual a **HabitacionInicioID**.
6. Mostrarle el siguiente mensaje: "$AVATAR, te encuentras en el cuarto $NOMBRE. $DESCRIPCION", donde $AVATAR es el nombre ingresado por el jugador y $NOMBRE y $DESCRIPCION son el nombre y la descripción de la habitación, respectivamente.
7. Si hubiera uno o más enemigos en la habitación, se le deberá informar mediante el siguiente mensaje: "En la habitación se encuentra un $ENEMIGO, el cual no está dispuesto a dejarte pasar. Su cantidad de puntos de vida es $HP.", siendo $ENEMIGO el nombre del enemigo y $HP sus puntos de vida. Los puntos de vida se calculan con un número pseudo-aleatorio entre **MinHP** y **MaxHP**. Seguido a esto, se deberá simular el combate (ver sección **Simulación de Combate**). Si hubiera más de un enemigo en la habitación, el jugador procederá a combatir con cada uno de ellos en forma secuencial. Nótese que el programa debe recordar si se han eliminado o no los enemigos de una habitación, es decir, si el *Avatar* vuelve a visitar una habitación donde inicialmente hubo enemigos no debe realizarse nuevamente el proceso de simulación de combate.
8. Si no hubiera enemigos en la habitación o éstos han sido derrotados, existen dos opciones:
    a. Si el identificador de la habitación actual corresponde al de la habitación de salida, informarle al jugador que ha completado el juego, permitiéndole salvar las acciones realizadas durante el juego (ver comando **dumpActions** en la sección **Diseño e implementación del programa**). Seguido a esto, se deberá terminar el programa y devolver el control al sistema operativo.
    b. Caso contrario, se le deberá informar sobre las puertas que allí se encuentran mediante un número entre **1** y **número de puertas disponibles** y darle la posibilidad de elegir una. Volver al paso 6.

4. Simulación de combate

La simulación del combate consiste en los siguientes pasos:

1. Obtener un número pseudo-aleatorio de punto flotante en el intervalo [0,1). Si el número es menor que 0.5, el jugador realiza el primer ataque. Caso contrario, es el enemigo quien comienza el ataque. Imprimir un mensaje indicando quién de los dos inicia el ataque.
2. Obtener un número pseudo-aleatorio entero en el intervalo [**MinDP**,**MaxDP**], donde **MinDP** y **MaxDP** son los atributos del atacante. Nótese que si el atacante es un enemigo, los valores de **MinDP** y **MaxDP** dependen del tipo de enemigo y de la profesión del *Avatar* que está atacando (para ello, se debe consultar la matriz creada dinámicamente al principio del programa). Imprimir un mensaje indicando los puntos de daño del golpe y pedirle al usuario que presione una tecla para continuar.
3. Restar el número aleatorio obtenido a los **HP** del adversario. Imprimir un mensaje indicando los **HP** restantes del adversario.
4. Si los **HP** del adversario son menores o iguales a cero, se finaliza el combate y la victoria corresponde al atacante, imprimiendo además un mensaje que indique quién fue el vencedor. Caso contrario, repetir los pasos 2, 3 y 4 intercambiando los roles. Es decir, el atacante pasa a ser atacado y viceversa.

5. Diseño e implementación del programa

Se debe realizar un diseño donde se separe claramente la interfaz con el usuario (front-end) del procesamiento de los datos del juego (back-end). Esto se verá reflejado en la implementación de una biblioteca de funciones de back-end en el archivo `dungeonCrawlerBack.c` y otro conjunto de funciones de front-end que invocan a la biblioteca. Estas últimas incluyen a la función main y corresponden al archivo `dungeonCrawlerFront.c`.

En ningún caso se debe repetir código para resolver situaciones similares, sino que debe implementarse una correcta modularización y se deben reutilizar funciones parametrizadas.

Tanto la biblioteca como el front-end deben estar correctamente comentados y en el caso de la biblioteca se debe escribir el archivo de encabezado correspondiente.

El programa deberá poder aceptar por línea de comandos dos parámetros extra:

- El primero indica el nombre archivo de donde debe ser tomada la información para el juego.
- El segundo, que es opcional, indica el número entero que corresponde a la semilla con la que se deberá inicializar el generador de números pseudo-aleatorios. Este parámetro será utilizado sólo con el fin de depurar el programa. Si no se especificara el parámetro extra al ejecutar el programa por línea de comandos, se deberá inicializar el generador de números pseudo-aleatorios utilizando la hora actual.

La implementación del front-end del juego es libre. La única restricción es que, a partir del momento en que el *Avatar* es situado en la habitación de inicio, el usuario pueda en cualquier momento de la partida ingresar por entrada estándar el siguiente comando:

> **dumpActions** arch.txt

Siendo arch.txt el nombre del archivo de texto de salida. El comando **dumpActions** volcará en el archivo de salida un informe que indique el valor con el cual fue inicializado el generador de números pseudo-aleatorios seguido de la secuencia en que fueron visitadas cada una de las habitaciones del laberinto junto con el número de puerta elegida en cada una de ellas. Nótese que NO deben asumir que el número máximo de habitaciones visitadas a lo largo de una partida tiene una cota máxima.

Ante una operación inválida (división por cero, segmentation fault, etc), el programa debe mostrar un mensaje adecuado pero nunca abortar.

ALGUNOS CONSEJOS:

No escribir el programa entero y después probarlo todo junto, sino escribir cada función, probándola por separado. Programar defensivamente en todos los casos.

Escribir el esquema de cada función primero en papel, pudiendo utilizar pseudocódigo o lenguaje coloquial.

Una buena metodología es comenzar a escribir las funciones de más alto nivel, *cableando* las inferiores con *cuerpo nulo* o con un *valor fijo*, e ir reemplazándolas de a una por vez, en la medida en que al integrarlas todo siga funcionando correctamente. Esto es implementación **Top-Down**.

Otra metodología es comenzar a escribir las funciones desde el nivel inferior, una por vez, probando a cada una con un pequeño main que sólo la invoque a ella. Una vez escritas y verificadas todas las funciones, unificarlas en un solo módulo. Esto es implementación **Bottom-Up.**

6. Material a entregar.

Cada grupo deberá entregar en sobre manila, con el nombre de los integrantes en el frente, el siguiente material:

- Impresión de todos los códigos fuente.
- Impresión del árbol de funciones (como los presentados en el TP Nro. 7)

A su vez, cada grupo deberá enviar un archivo tipo zip a la cuenta `pi@it.itba.edu.ar` con los siguientes archivos:

- `dungeonCrawlerFront.c`
- `dungeonCrawlerBack.h`
- `dungeonCrawlerBack.c`
- `makefile`
- Otros programas fuentes (de ser necesario)
- Al menos un archivo de texto con la definición de un juego que cuente por lo menos con cinco tipos de enemigos distintos, tres profesiones y quince habitaciones.
- Un mapa que indique, para el archivo de texto del inciso anterior, la disposición de las habitaciones, las conexiones existentes entre ellas y los enemigos que las habitan.
- Un archivo de texto con la salida generada con el comando **dumpActions** que indique, para el mapa creado, la semilla del generador de números pseudo-aleatorios y el orden en el que se deben recorrer las habitaciones y puertas para obtener una partida victoriosa.
- **Puntos extra:** Implementar alguna característica nueva para el juego. Algunas ideas: pociones mágicas que recuperen algunos HP del personaje, posibilidad de encontrar nuevas armas y usarlas, etc.

La impresión de los códigos debe hacerse de forma tal que el código pueda leerse fácilmente.

Ejemplos a no seguir:

Código mal indentado

```
for(i=0; i < filas; i++)
for(j=0; i < columnas; i++)
count = …
```

Líneas que ocupan más del ancho de página y continúan en otra línea

```
while ( … )
{
     for ( i = 0; i < filas; i ++ )  /*  en  este  ciclo  vamos  a
verificar si hay posiciones libres */
     {
          .....
```

Comentarios innecesarios

```
i++; // Incrementamos la variable i en 1
```

7. Fecha de entrega.

El trabajo debe entregarse por e-mail el Viernes 4 de Junio hasta las 20 hs (todos los e-mails recibirán confirmación de lectura), los materiales impresos se deben entregar el Lunes 7 de Junio antes de comenzar la clase.

Aquel grupo que entregue el trabajo **en fecha** y resulte desaprobado, tendrá la oportunidad de recuperarlo. Si el recuperatorio resultara satisfactorio, la nota del mismo será de 4 puntos.

Por otra parte, los alumnos pueden optar por entregar tarde (hasta una semana después de la fecha prevista), pero en ese caso **se le restarán dos puntos** a la nota del trabajo y **no tendrán posibilidad de recuperatorio**.

8. Criterios de Evaluación y Calificación

Para la evaluación y calificación del trabajo especial se considerarán:

- el correcto funcionamiento del programa (recordar el uso de métodos de prueba de software).
- la modularización realizada.
- la claridad del código.
- el cumplimiento de las reglas de estilo de programación dadas en clase.
- la presentación del trabajo en todos los aspectos (documentación, facilidad de uso, etc).

9. Consultas

Cualquier consulta que se desee realizar sobre el enunciado o la implementación del trabajo especial deberá hacerse en el horario propio de consulta de la materia o por e-mail a pi@it.itba.edu.ar. En caso de hacer la consulta vía e-mail, escribir el prefijo **[TPE]** en el nombre del mensaje.