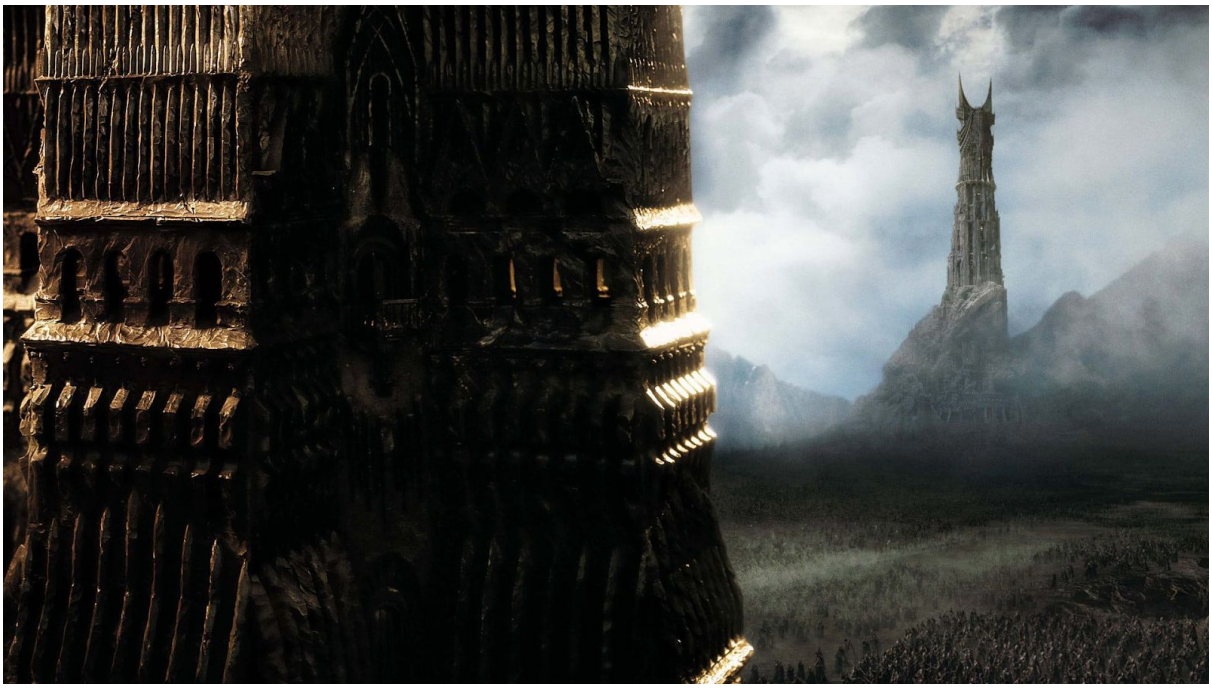
Algoritmos y Programación I
Curso Mendez

.UBAfiuba
FACULTAD DE INGENIERÍA

Defendiendo las Torres

TP3



Fecha de Presentación: 16/07/2020

Fecha de Vencimiento: 30/07/2020

1. Introducción

El Abismo de Helm era un desfiladero que se abría paso entre las Ered Nimrais bajo el Thrihyrne. El Abismo de Helm era el centro defensivo del Folde Oeste de Rohan; éste y El Sagrario eran las principales fortalezas y refugios del reino. La Corriente del Bajo salía del Abismo de Helm, y allí se hallaban las Aglarond. El Muro del Bajo se levantó para cerrar la entrada al Abismo.

Durante la Guerra del Anillo los hombres de Rohan se refugiaron en el Abismo de Helm del asedio del ejército de Saruman el Blanco, formado por enormes fuerzas, más de 10.000 Uruk-Hai, orcos, semiorcos y huargos.

El objetivo de las fuerzas de Isengard era aniquilar totalmente al pueblo de Rohan de un golpe, y dejar a Gondor sin aliados formales ante las legiones de Sauron.

Algunos héroes conocidos de esta batalla son Aragorn, Legolas y Charly, además de Théoden, rey de Rohan y Éomer su sobrino y heredero.

Las huestes de Saruman llegaron al Valle del Abismo de Helm en medio de la noche, y muy rápido escalaron la primera defensa e intentaron derribar la puerta de la fortaleza con un ariete. Pero Aragorn, Éomer y algunos otros rohirrim atacaron, a través de una puerta trasera en el lado de Horburg, dispersando a los orcos que atacaban las puertas.

Los orcos y los dunledinos levantaron entonces cientos de escaleras para escalar el muro de la fortaleza. Aragorn y Éomer tuvieron que mover repetidas veces a los defensores, que estaban agotados, para poder repeler a los orcos, subiendo escaleras, y cruzando el muro de un lado a otro. Sin embargo, algunos orcos pudieron infiltrarse a través de una pequeña alcantarilla, que permitía que una corriente de agua saliese del Abismo de Helm, y mientras los rohirrim estaban repeliendo a los orcos que escalaban el muro, intentaron llegar a ella luego de superar una barrera. Los defensores reaccionaron rápidamente y expulsaron a los orcos, y la alcantarilla fue bloqueada bajo la supervisión de Charly.

Pero los orcos volvieron a entrar en la alcantarilla, y gracias a un dispositivo ideado por Saruman, causaron una gran explosión, creando un enorme agujero en el muro del Abismo.

Los orcos pudieron entrar sin poder ser detenidos. Los defensores se retiraron a las Cuevas Relucientes y al Hornburg. Pronto las fuerzas de Saruman usaron sus armas para ganar la entrada a la fortaleza, que antes había defendido Aragorn.

La batalla no ha terminado aún, es momento de tomar partida, para uno u otro bando...

2. Objetivo

El presente trabajo práctico tiene como objetivo que el alumno:

- Se familiarice con el manejo de archivos en C.
- Utilice las funciones acordes al tipo de archivo y/o forma de acceso.
- Realice apropiadamente las operaciones con archivos.
- Ponga en práctica el uso de argumentos por línea de comando.

Por supuesto, se requiere que el trabajo cumpla con las buenas prácticas de programación profesadas por la cátedra.

Se considerarán críticos la modularización, reutilización y claridad del código.

3. Enunciado

Desde la alta dirección de Las 2 Torres, se desea ampliar la funcionalidad del programa creado, incluyendo características que permitan llevar registro de las partidas jugadas.

Se pide, entonces, amoldar el trabajo desarrollado para cumplir con los siguientes requerimientos:

- **Mostrar ranking:** Deberá poder mostrar el ranking de los participantes, ordenado de mayor a menor según el puntaje obtenido en la partida. Dicho ranking corresponderá a cierta configuración de juego.
- **Crear caminos:** Los jugadores podrán crear caminos para utilizar luego durante las partidas.

- **Crear configuraciones de juego:** Se podrán configurar ciertas características del juego, brindando una mayor flexibilidad a las partidas.
- **Ver partida grabada:** Las repeticiones de las partidas grabadas estarán disponibles para ser reproducidas en cualquier momento.
- **Jugar partidas personalizadas:** Tomando alguna de las configuraciones creadas el jugador podrá jugar una partida personalizada, así como elegir grabarla para verla luego.

4. Especificaciones

El trabajo a realizar no se basará en un menú, sino que sus funcionalidades serán ingresadas por línea de comando.

Algunos de los parámetros pueden no ingresarse, por lo que se ha tomado la convención de que en aquellos comandos en los que sus parámetros son opcionales, se los acompañará de una etiqueta que hace referencia a aquello que representan.

4.1. Mostrar Ranking

Listará todos los jugadores con los puntajes obtenidos al terminar las partidas.

Habrá un ranking por cada configuración ya que dependiendo de ella variará la dificultad del juego.

Queda a criterio del alumno como y que información visualizar en el listado.

4.1.1. Comando

Recibirá como parámetro la cantidad de jugadores a mostrar y el archivo de configuración para el cual se quiere ver el ranking

Un ejemplo de este comando es:

```
./defendiendo ranking listar=25 config=mi_configuración.txt
```

En el caso del ejemplo, se mostrarán los primeros 25 jugadores del ranking para la configuración mi_configuracion.txt.

En caso de no recibir la cantidad de jugadores a mostrar, se mostrarán todos.

En caso de no recibir el archivo de configuración se mostrará el ranking de la configuración por defecto.

Ambos parámetros son opcionales.

4.1.2. Archivos de ranking

Será un archivo de texto cuyos valores estarán separados por ;.

El nombre de cada archivo de ranking estará formado por la palabra ranking_ luego seguirá el nombre del archivo de configuración y tendrán extensión csv.

Es decir, si en una partida se utiliza el archivo de configuración con nombre una_configuracion.txt, el puntaje será reflejado en ranking_una_configuracion.csv

En caso de que la partida se juegue con la configuración default, el puntaje será guardado en ranking.csv

Los campos de cada línea serán:

```
nombre del jugador;puntaje obtenido
```

El puntaje de cada jugador está dado por la siguiente fórmula:

$$Puntaje = \frac{orcos\_muertos * 1000}{torre\_1 + torre\_2 + enanos\_inicio + elfos\_inicio + enanos\_extra + elfos\_extra}$$

Siendo orcos_muertos la cantidad de orcos muertos en la partida, torre_1 la resistencia inicial de la torre 1, torre_2 la resistencia inicial de la torre 2, enanos_inicio la cantidad total de enanos que se utilizan al iniciar los niveles, elfos_inicio

la cantidad total de elfos que se utilizan al iniciar los niveles, enanos_extra la cantidad de enanos extra que se pueden utilizar y elfos_extra la cantidad de elfos extra que se pueden utilizar.

Ésta fórmula es brindada por la cátedra e intenta penalizar el uso de mayores recursos. Es totalmente válida la creación de un cálculo de puntaje propio siempre que se pondere el uso de los recursos.

El puntaje será un número entero, en caso de haber empate en puntos, esos jugadores quedarán ordenados alfabéticamente por nombre.

Un ejemplo de este archivo se muestra a continuación:

```
Fracca;2500
Santi;2499
...
Charly;3
```

4.2. Crear caminos

Permitirá crear los caminos para utilizar durante las partidas.

Se le debe permitir al usuario crear los caminos de forma sencilla, mostrándole por pantalla como va quedando el camino creado.

Es imprescindible corroborar que cada camino está formado correctamente, es decir que las coordenadas en el orden dispuesto forman un camino válido, sin quedar coordenadas desconectadas.

Asimismo, los caminos deben crearse para todos los niveles.

Queda a criterio del alumno como mostrar y solicitar la información de los caminos que se están creando.

4.2.1. Comando crear_camino

Recibirá como parámetro el nombre del archivo.

Le permitirá al jugador crear los caminos para utilizar luego.

Un ejemplo de este comando es:

```
./defendiendo crear_camino mis_caminos.txt
```

En caso de no ingresar nombre de archivo, el comando no será válido y no se realizará ninguna acción.

En caso de existir un archivo de caminos con ese nombre, el archivo se sobreescribirá.

El parámetro es obligatorio.

4.2.2. Archivos de caminos

Será un archivo de texto cuyos valores estarán separados por ; y de extensión txt.

Este archivo contendrá información de todos los caminos de una partida.

Los archivos de caminos deberán estar formados por una cabecera que indica a que nivel corresponden las coordenadas, luego a que camino corresponden y luego las coordenadas.

Un ejemplo de este archivo se muestra a continuación:

```
NIVEL=1
CAMINO=1
8;14
8;13
7;13
...
10;0
NIVEL=2
CAMINO=1
3;0
...
14;14
NIVEL=3
```

```
CAMINO=1
0;6
...
CAMINO=2
0;6
...
```

4.3. Crear Configuración Inicial

Los archivos de configuración permitirán jugar con mayor flexibilidad.

Queda a criterio del alumno como mostrar y solicitar la información de las características configurables, así como la validación de las mismas.

En características que no se deseen configurar se les asignará el valor -1.

Se deben poder configurar las siguientes características del juego:

- Resistencia de las torres:

```
RESISTENCIA_TORRES=<puntos de la torre 1>,<puntos de la torre 2>
```

- Cantidad de enanos al iniciar cada nivel:

```
ENANOS_INICIO=<enanos nivel 1>,<enanos nivel 2>,<enanos nivel 3>,<enanos nivel 4>
```

- Cantidad de elfos al iniciar cada nivel:

```
ELFOS_INICIO=<elfos nivel 1>,<elfos nivel 2>,<elfos nivel 3>,<elfos nivel 4>
```

- Cantidad de enanos extra y el costo para cada torre:

```
ENANOS_EXTRA=<cantidad de enanos extra>,<costo a la torre 1>,<costo a la torre 2>
```

- Cantidad de elfos extra y el costo para cada torre:

```
ELFOS_EXTRA=<cantidad de elfos extra>,<costo a la torre 1>,<costo a la torre 2>
```

- Ánimo de los enanos:

```
ENANOS_ANIMO=<Fallo de los enanos>,<Crítico de los enanos>
```

- Ánimo de los elfos:

```
ELFOS_ANIMO=<Fallo de los elfos>,<Crítico de los elfos>
```

- Velocidad del juego:

```
VELOCIDAD=<Tiempo que se detendrá cada turno (float)>
```

- Caminos:

```
CAMINOS=<Ruta del archivo de caminos a utilizar en esta configuación>
```

4.3.1. Comando crear_configuracion

Recibirá como parámetro el nombre del archivo.

Le permitirá al jugador crear una configuración inicial para utilizar luego.

Un ejemplo de este comando es:

```
./defendiendo crear_configuracion mi_configuracion.txt
```

En caso de no ingresar nombre de archivo, el comando no será válido y no se realizará ninguna acción.

En caso de existir un archivo de configuración con ese nombre, el archivo se sobreescribirá.

El parámetro es obligatorio.

Se deberán solicitar al usuario todos los valores, si no desea cargar alguno de ellos, se almacenará un -1 y se utilizará, para esa característica, el valor por defecto del juego.

4.3.2. Archivos de configuración

Será un archivo de texto con palabras clave que indican a que característica refieren.

Los valores de este archivo se da por entendido que son válidos y están todos.

El orden de las etiquetas no es fijo, por lo se deberá buscar cada una de ellas a la hora de utilizarlas.

Un ejemplo de este archivo se muestra a continuación:

```
RESISTENCIA_TORRES=1000,1000
ENANOS_INICIO=8,0,-1,4
ELFOS_INICIO=0,6,-1,2
ENANOS_EXTRA=5,75,-1
ELFOS_EXTRA=5,-1,100
ENANOS_ANIMO=30,20
ELFOS_ANIMO=60,15
VELOCIDAD=0.4
CAMINOS=-1
```

En el ejemplo, la resistencia de cada una de las torres es 1000, los enanos al inicio del nivel 3 deben ser tomados por defecto, los caminos no han sido cargados por lo que deberá usarse la forma por defecto para generarlos, entre otras cosas.

4.4. Ver partida grabada

Le permitirá a los usuarios, ver una partida grabada anteriormente, la visualización es automática y no deberá solicitarle nada al usuario.

4.4.1. Comando poneme_la_repe

Recibirá como parámetro el nombre del archivo con la partida y la velocidad de reproducción.

Un ejemplo de este comando es:

```
./defendiendo poneme_la_repe grabacion=mi_partida.dat velocidad=0.5
```

En caso de no ingresar nombre de archivo, el comando no será válido y no se realizará ninguna acción.

En caso de no ingresar velocidad de reproducción, por defecto se tomará velocidad 1.

El parámetro con el archivo de la grabación es obligatorio, la velocidad es opcional.

4.4.2. Jugar partida

Permitirá jugar una partida utilizando configuraciones creadas por los usuarios. Adicionalmente el jugador podrá guardar la partida para verla luego.

4.4.3. Comando jugar

Recibirá como parámetros el nombre del archivo de caminos a utilizar y, en caso de querer grabar la partida, el nombre del archivo donde se grabará la partida.

Un ejemplo de este comando es:

```
./defendiendo jugar config=mi_configuracion.txt grabacion=mi_partida.dat
```

Si el archivo de configuración no existe o no fue ingresado, debe utilizarse la configuración por defecto.

Si el archivo de grabación ya existe, se sobreescribirá, si el parámetro grabación no fue ingresado, la partida no se grabará.

Ambos parámetros son opcionales.

4.4.4. Archivos de partidas

Será un archivo binario de acceso secuencial donde cada registro contiene un turno de la partida, desde que comienza hasta que termina.

Los estados del juego en cada turno se entiende que son válidos.

5. Cambios en el TP2

La idea de este trabajo es que los cambios que sufra la biblioteca sean los menos posible.

Es así que se deberá crear una biblioteca extra, donde se desarrollara toda la funcionalidad pedida.

Asimismo es necesario agregar 1 parámetro en la función inicializar_juego para que reciba la configuración a inicializar.

De esta manera quedaría:

```
void inicializar_juego(..., configuracion_t configuracion);
```

La creación de la estructura queda a criterio del alumno, y su utilización permitirá inicializar todas las características configurables en el inicio de la partida.

6. Compilación y Entrega

El trabajo debe poder ser compilado correctamente con la siguiente línea:

```
gcc *.c -o defendiendo -std=c99 -Wall -Werror -Wconversion
```

Se deben cumplir todas las pautas mencionadas y deben funcionar todas las opciones. Además debe ser entregado por la plataforma kwyjibo y se tendrá en cuenta la legibilidad del código y las buenas prácticas al igual que en los trabajos anteriores.

Al poner *.c el compilador toma todos los .c de la carpeta y los compila. Por ende, no importa el nombre que le pongan a la biblioteca ni al programa. Ante cualquier eventualidad informaremos oportunamente.

7. Anexo

7.1. Argumentos por línea de comando

Para poder pasar parámetros a un programa a través de la línea de comandos nos valemos de la siguiente declaración de la función main:

```
int main(int argc, char *argv[]){..
```

Argc contiene la cantidad de argumentos recibidos por el programa, debemos considerar que siempre será el número de argumentos pasados más 1, ya que el primer argumento se reserva para contener el nombre del programa. **Argv** es un vector de strings que contiene los parámetros pasados en el mismo orden en que fueron escritos.

El siguiente programa muestra un ejemplo de cómo hacer uso de estos parámetros:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

        printf ("El programa recibió %i argumentos\n", argc);
        for(int i = 0; i < argc; i++) {
                printf("Parámetro %d: %s\n", i, argv[i]);
        }

        if (strcmp (argv[1], "saludar") == 0){
                printf ("Hola!\n");
        } else if (strcmp (argv[1], "sumar") == 0 && argc == 4){
```

```
15                  int sumando_1 =  atoi (argv [2]);
16                  int sumando_2 =  atoi (argv [3]);
17                  printf ("%i + %i = %i\n", sumando_1, sumando_2, sumando_1 + sumando_2);
18          }
19          return 0;
20  }
```

Una vez compilado, sólo nos resta ingresar la línea de comando según lo que se quiera hacer, si ingresamos:

```
1  ./ejemplo sumar 20 54
```

Siendo ejemplo el nombre del programa, lo que se muestra es:

```
1  El programa recibió 4 argumentos
2  Parámetro 0: ./ejemplo
3  Parámetro 1: sumar
4  Parámetro 2: 20
5  Parámetro 3: 54
6  20 + 54 = 74
```

7.2. Atoi: Array to integer

```
1  int atoi(const char *nptr);
```

Esta función sirve para convertir la porción inicial de una cadena de caracteres apuntada por **nptr** en un entero.

Para usarla debemos incluir la biblioteca **stdlib.h**.

7.3. Atof: Array to double

```
1  double atof(const char *nptr);
```

Esta función sirve para convertir la porción inicial de una cadena de caracteres apuntada por **nptr** en un double.

En el caso del presente trabajo es necesario convertir una cadena de caracteres a un float, lo cual puede realizarse de la siguiente manera:

```
1  float numero = (float) atof(cadena_de_caracteres);
```

Para usarla debemos incluir la biblioteca **stdlib.h**.