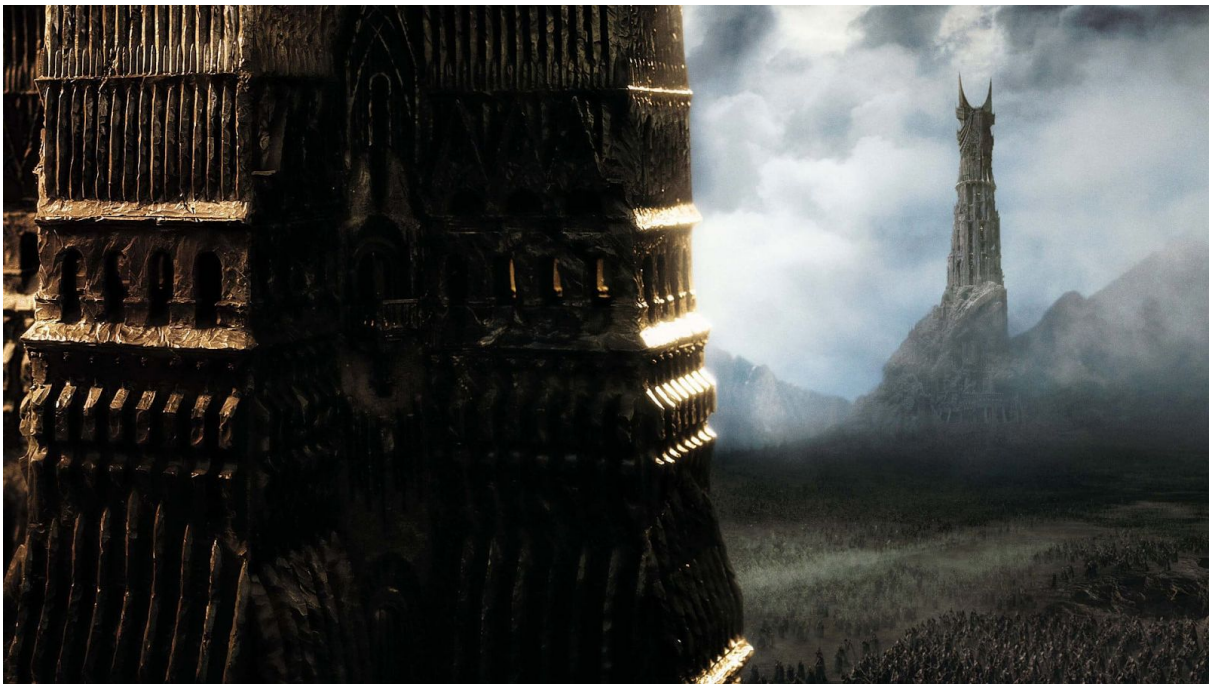
Algoritmos y Programación I
Curso Mendez

.UBAfiuba
FACULTAD DE INGENIERÍA

Defendiendo las Torres

TP2



Fecha de Presentación: 21/05/2020

Fecha de Vencimiento: 25/06/2020

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:

- Diseñe y desarrolle las funcionalidades de una biblioteca con un contrato preestablecido.
- Se familiarice con y utilice correctamente los tipos de datos estructurados.
- Desarrolle una interfaz gráfica amigable y entendible para el usuario.

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

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

3. Enunciado

Luego de un intenso trabajo de estrategia e ingeniería se lograron aislar Las 2 Torres, dejando solo unos pocos caminos que conducen a ellas por cada uno de los puntos cardinales.

- **Entrada Este**: Posee un camino que conduce a la Torre 1, la defensa fue asignada a Gimli y su robusto equipo de enanos para defenderlo.
- **Entrada Oeste**: Posee un camino que conduce a la Torre 2, la defensa fue asignada a Legolas y su ágil grupo de elfos.
- **Entrada Norte**: Posee dos caminos uno conduce a la Torre 1 y el otro a la Torre 2, se espera que tanto los enanos conducidos por Gimli como los elfos comandados por Legolas lleguen a tiempo para el ataque.

- **Entrada Sur**: Posee, al igual que la entrada Norte, dos caminos uno conduce a la Torre 1 y el otro a la Torre 2, se espera que tanto los enanos conducidos por Gimli como los elfos comandados por Legolas lleguen a tiempo para el ataque.

Se deberá desarrollar un programa que permita Defender las 2 Torres de la embestida enemiga.

Según fuentes oficiales, los orcos tienen la orden de no detenerse a luchar, de avanzar sin distracciones hasta las torres.

El juego se desarrollará en 4 niveles, cada uno de ellos representa el ataque por cada una de las entradas en el orden Este (Nivel 1), Oeste (Nivel 2), Norte (Nivel 3) y Sur (Nivel 4), en todos ellos una horda de orcos intentará derribar la o las torres presentes.

Para evitarlo, el jugador deberá posicionar enanos y elfos en las inmediaciones del recorrido de la horda.

La partida terminará victoriosa, si ninguna de las 2 torres es destruida, asimismo, con que una de ellas caiga, se considerará una derrota.

3.1. Enanos

El peso del hacha permite golpear muy fuerte a los enemigos, pero solo de a 1 por turno.

- **Rango de ataque**: todos los casilleros alrededor de donde se encuentra el defensor.
- **Fuerza de ataque**: 60 puntos.
- **Ataque crítico**: 100 puntos. Depende del ánimo de Gimli.
- **Fallo**: Depende de la humedad.

Algunos de los ataques de los enanos serán críticos, la probabilidad de que ésto ocurra depende del ánimo de Gimli. Si su ánimo es malo, no habrá ataques críticos, si el ánimo es regular, el 10 % de los ataques serán críticos y si el ánimo es bueno, el 25 % de los ataques serán críticos.

Algunos de los ataques de los enanos serán fallidos, lo que implica que no le harán daño a ningún orco, y depende de la humedad del día. El % de fallo de cada hachazo de los enanos será de la mitad del valor de la humedad.

3.2. Elfos

Su agilidad les da la ventaja de atacar a todos los que están en su rango, aunque no siempre una flecha es letal. Atacan a todos los que están a 3 casilleros o menos en Distancia Manhattan (ver anexo).

- **Rango de ataque**: 3 casilleros de distancia.
- **Fuerza de ataque**: 30 puntos.
- **Ataque crítico**: 70 puntos. Depende del ánimo de Legolas.
- **Fallo**: Depende del viento.

Algunos de los ataques de los elfos serán críticos, la probabilidad de que ésto ocurra depende del ánimo de Legolas. Si su ánimo es malo, no habrá ataques críticos, si el ánimo es regular, el 10 % de los ataques serán críticos y si el ánimo es bueno, el 25 % de los ataques serán críticos.

Algunos de los ataques de los elfos serán fallidos, lo que implica que no le harán daño a ningún orco, y depende del viento del día. El % de fallo de cada flechazo de los elfos será de la mitad del valor de la velocidad del viento.

Es pertinente aclarar que, en el caso de los elfos que atacan a varios enemigos a la vez, la probabilidad de fallo y criticidad es ante cada uno de los orcos, y no del turno, es decir, si un elfo tiene a 2 orcos dentro de su rango para atacar en un determinado turno, es posible que a uno lo ataque y al otro no.

3.3. Orcos

Los orcos no atacaran ni a los enanos ni a los elfos, solo caminarán por el camino establecido hasta llegar a la torre que dicho camino los conduce, o morir en el intento.

Su vida es de 200 puntos + un extra aleatorio entre 0 y 100.

3.4. Torres

Las torres se encontrarán en el último casillero de un camino.

Cada una de las torres tiene una resistencia de 600 puntos.

Si un orco llega a ella, le resta tantos puntos de resistencia como puntos de vida tenga ese orco.

Se dice que la torre cayó, cuando su resistencia llega a 0.

La resistencia de cada una de las torres no se reiniciará al cambiar de nivel.

Las torres cuentan con 10 enanos y 10 elfos extras, para ser usados durante la partida, éstos no podrán usarse si al elegirlos la resistencia de la torre correspondiente llegara a 0.

Cada enano extra usado le cuesta 50 puntos de resistencia a la Torre 1.

Cada elfo extra usado le cuesta 50 puntos de resistencia a la Torre 2.

3.5. Niveles

A las torres se puede acceder por los 4 puntos cardinales, sin embargo, cada uno de ellos tiene ciertas características particulares que se explican a continuación.

3.5.1. Este

La entrada del Este tiene 1 solo camino que lleva a la Torre 1, el mismo se recorre de derecha a izquierda y será defendido inicialmente por 5 enanos.

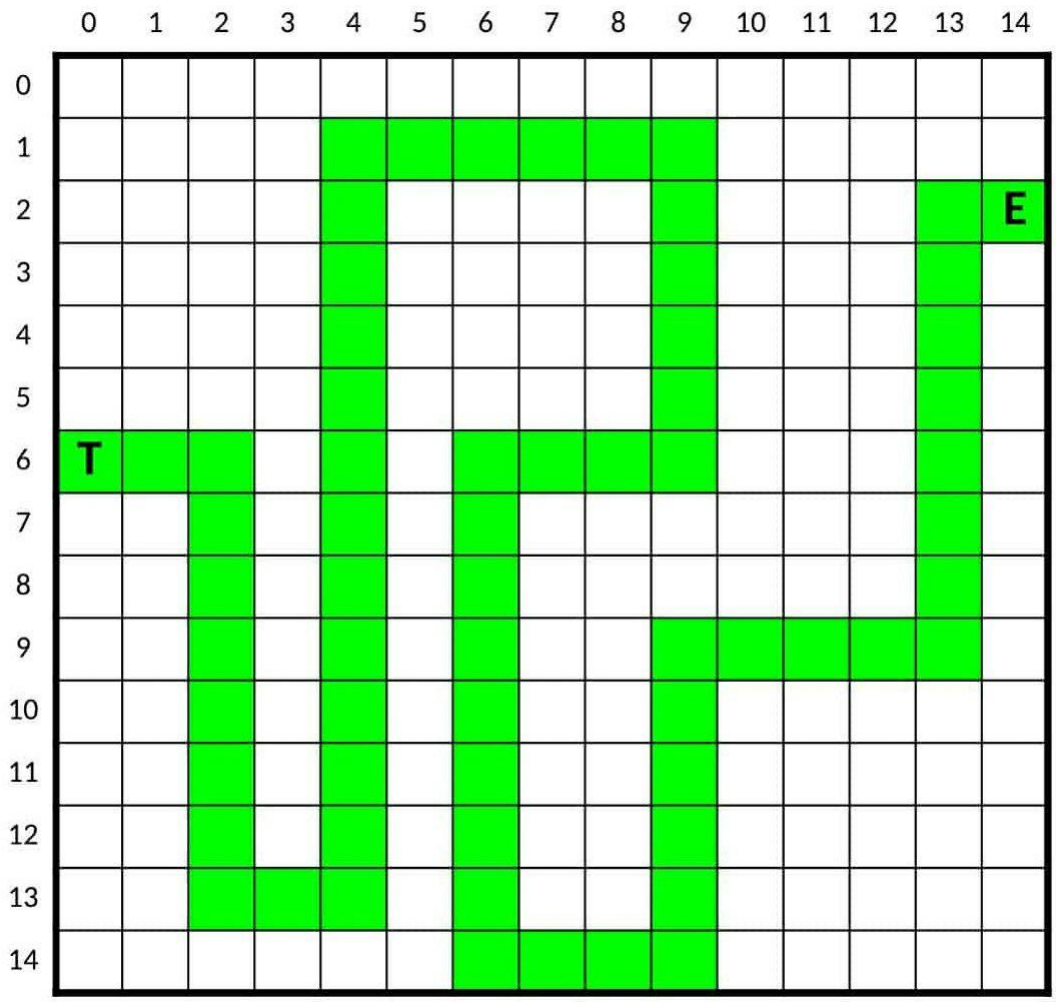
Los mismos debes serán posicionados al comenzar el nivel, el alumno debe decidir como hacerlo, si le deja al usuario elegir las posiciones, si se posicionan aleatoriamente o una combinación de ellos.

Los defensores no podrán ser reubicados y la posición no puede coincidir con la de algún casillero del camino, ni donde haya otro enano, ni donde haya otro elfo.

Se sabe que la horda que entrará por el Este constará de 100 orcos.

Cada 25 orcos, podrá posicionarse 1 enano extra si el jugador lo desea.

Este nivel debe representarse como un tablero de 15x15 y un recorrido posible de la horda se muestra a continuación.



3.5.2. Oeste

La entrada del Oeste tiene 1 solo camino que lleva a la Torre 2, el mismo se recorre de izquierda a derecha y será defendido inicialmente por 5 elfos.

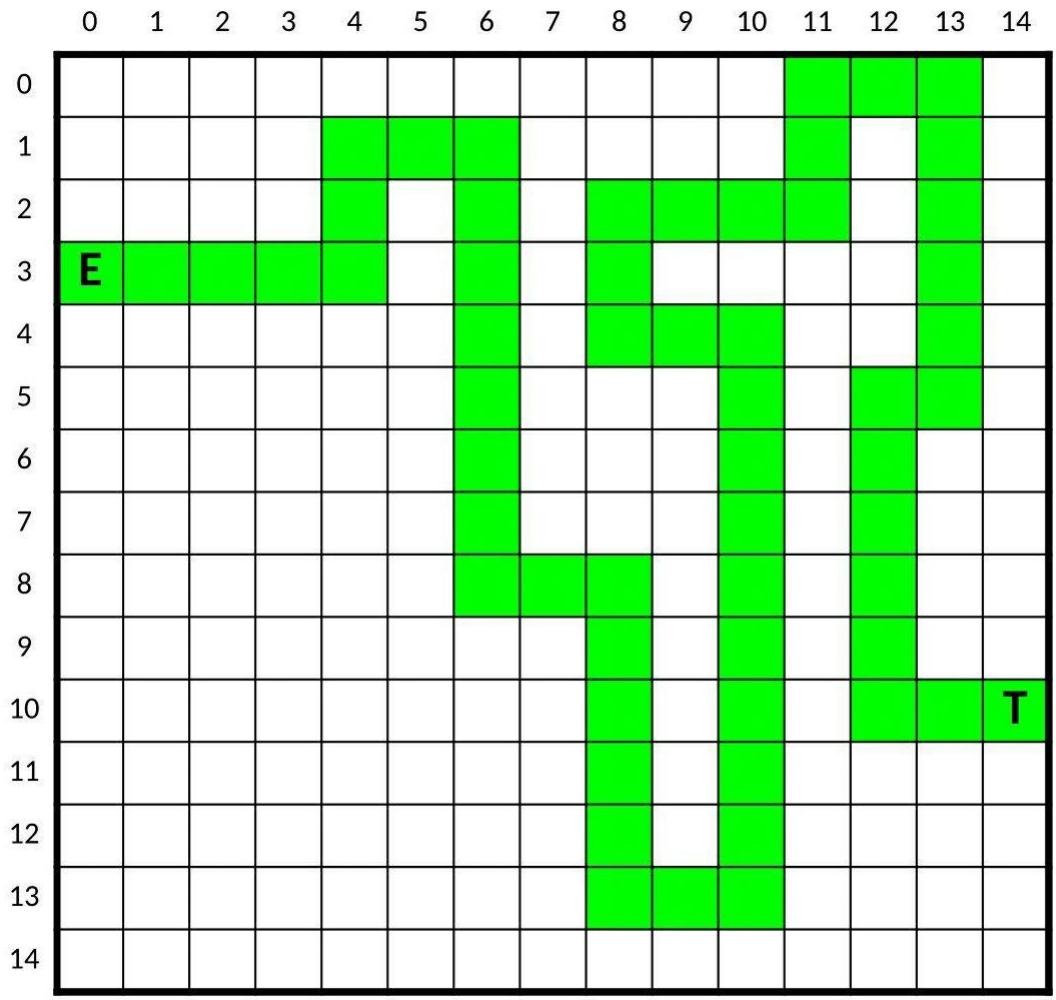
Los mismos debes serán posicionados al comenzar el nivel, el alumno debe decidir como hacerlo, si le deja al usuario elegir las posiciones, si se posicionan aleatoriamente o una combinación de ellos.

Los defensores no podrán ser reubicados y la posición no puede coincidir con la de algún casillero del camino, ni donde haya otro enano, ni donde haya otro elfo.

Se sabe que la horda que entrará por el Oeste constará de 200 orcos.

Cada 50 orcos, podrá posicionarse 1 elfo extra si el jugador lo desea.

Este nivel debe representarse como un tablero de 15x15 y un recorrido posible de la horda se muestra a continuación.



3.5.3. Norte

La entrada del Norte tiene 2 caminos, uno conduce a la Torre 1 y el otro a la Torre 2, ambos se recorren de arriba a abajo y serán defendidos inicialmente por 3 enanos y 3 elfos.

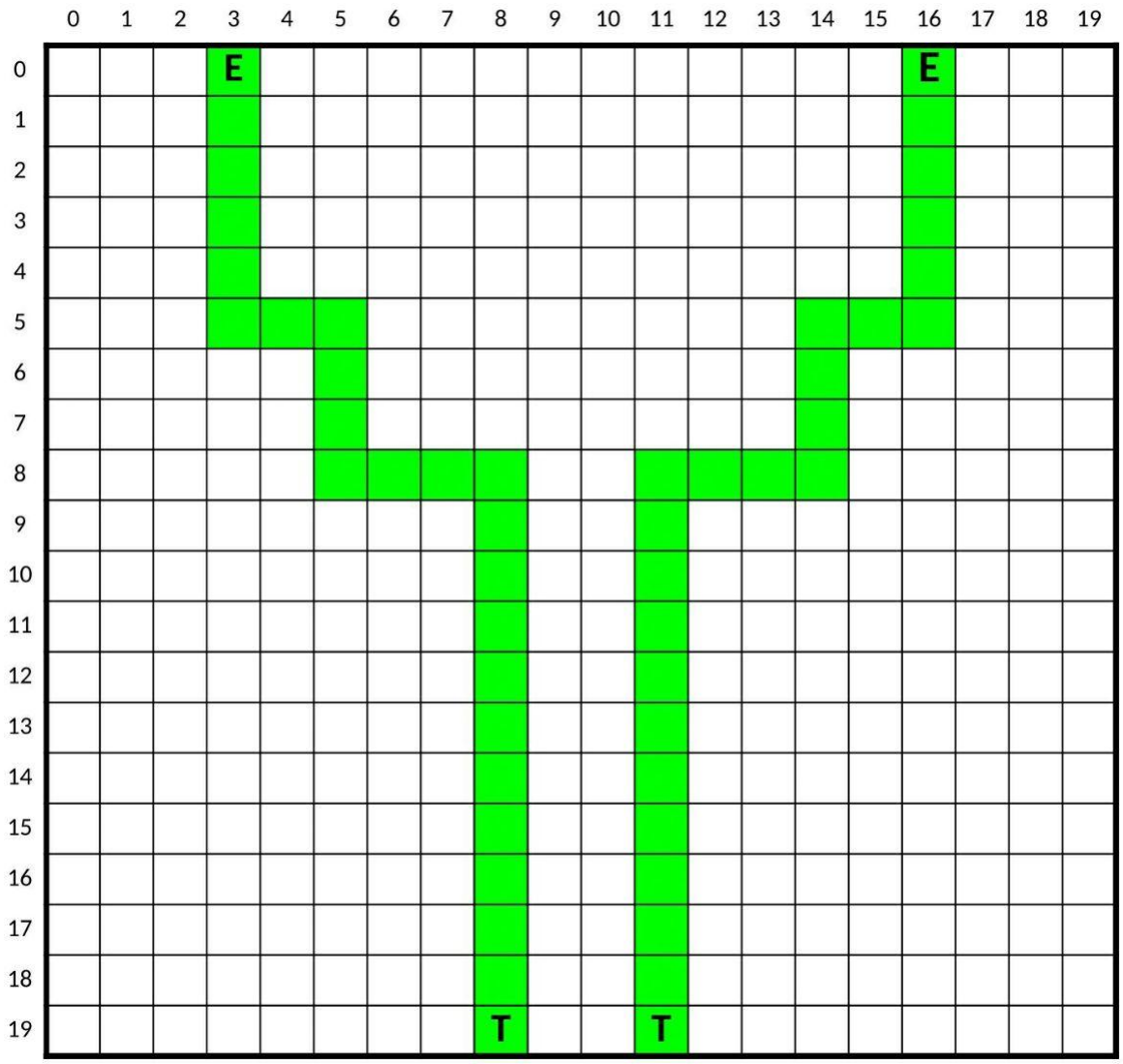
Los mismos debes serán posicionados al comenzar el nivel, el alumno debe decidir como hacerlo, si le deja al usuario elegir las posiciones, si se posicionan aleatoriamente o una combinación de ellos.

Los defensores no podrán ser reubicados y la posición no puede coincidir con la de algún casillero del camino, ni donde haya otro enano, ni donde haya otro elfo.

Se sabe que la horda que entrará por el Norte constará de 150 orcos por cada uno de los caminos.

Cada 50 orcos, podrá posicionarse un enano o un elfo extra si el jugador lo desea.

Este nivel debe representarse como un tablero de 20x20 y un recorrido posible de la horda se muestra a continuación.



3.5.4. Sur

La entrada del Sur tiene 2 caminos, uno conduce a la Torre 1 y el otro a la Torre 2, ambos se recorren de abajo a arriba y serán defendidos inicialmente por 4 enanos y 4 elfos.

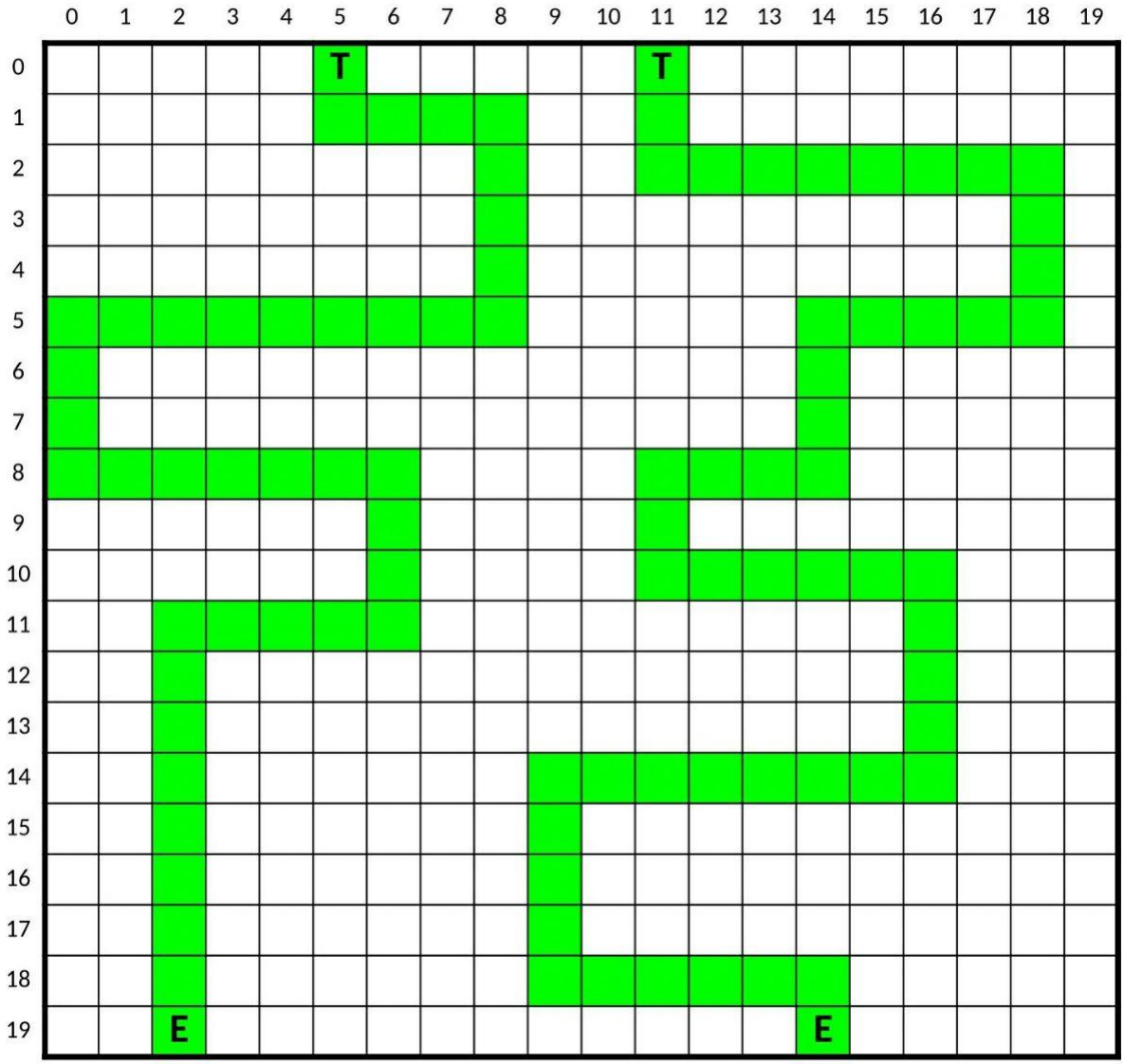
Los mismos debes serán posicionados al comenzar el nivel, el alumno debe decidir como hacerlo, si le deja al usuario elegir las posiciones, si se posicionan aleatoriamente o una combinación de ellos.

Los defensores no podrán ser reubicados y la posición no puede coincidir con la de algún casillero del camino, ni donde haya otro enano, ni donde haya otro elfo.

Se sabe que la horda que entrará por el Sur constará de 250 orcos por cada uno de los caminos.

Cada 50 orcos, podrá posicionarse un enano o un elfo extra si el jugador lo desea.

Este nivel debe representarse como un tablero de 20x20 y un recorrido posible de la horda se muestra a continuación.



4. Especificaciones

4.1. Biblioteca defendiendo_torres.h

A continuación se explicitan las estructuras a utilizar y las funciones públicas de la biblioteca a implementar.

Cabe destacar que pueden crearse y utilizarse todas las estructuras extra que se desee siempre y cuando se respeten las brindadas por la cátedra.

Asimismo de deberán crear todas las funciones privadas necesarias para obtener una correcta modularización del trabajo.

```
#ifndef __DEFENDIENDO_TORRES_H__
#define __DEFENDIENDO_TORRES_H__

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

#define MAX_FILAS 30
#define MAX_COLUMNAS 30
#define MAX_LONGITUD_CAMINO 200
#define MAX_ENEMIGOS 500
#define MAX_DEFENSORES 50

typedef struct coordenada {
```

```c
        int fil;
        int col;
} coordenada_t;

typedef struct defensor {
        char tipo;
        int fuerza_ataque;
        coordenada_t posicion;
} defensor_t;

typedef struct torres {
        int resistencia_torre_1;
        int resistencia_torre_2;
        int enanos_extra;
        int elfos_extra;
} torres_t;

typedef struct enemigo {
        int camino;
        int pos_en_camino;
        int vida;
} enemigo_t;

typedef struct nivel {
        coordenada_t camino_1[MAX_LONGITUD_CAMINO];
        int tope_camino_1;

        coordenada_t camino_2[MAX_LONGITUD_CAMINO];
        int tope_camino_2;

        defensor_t defensores[MAX_DEFENSORES];
        int tope_defensores;

        enemigo_t enemigos[MAX_ENEMIGOS];
        int tope_enemigos;
} nivel_t;

typedef struct juego {
        nivel_t nivel;
        torres_t torres;
        int nivel_actual;
        int critico_legolas;
        int critico_gimli;
        int fallo_legolas;
        int fallo_gimli;
} juego_t;

/*
 * Inicializará el juego, cargando la informacion de las torres y
 * los ataques críticos y fallo de Legolas y Gimli.
 * NO inicializará el primer nivel.
 */
void inicializar_juego(juego_t* juego, int viento, int humedad, char animo_legolas, char animo_gimli
    );

/*
 * Recibe un juego con todas sus estructuras válidas.
 * El juego se dará por ganado si el juego está en el ultimo nivel y éste ha sido terminado.
 * El juego se dará por perdido, si alguna de las torres llega a 0 en su resistencia.
 * Devolverá:
 * >  0 si el estado es jugando.
 * > -1 si el estado es perdido.
 * >  1 si el estado es ganado.
 */
int estado_juego(juego_t juego);

/*
 * Recibe un nivel con todas sus estructuras válidas.
 * El nivel se dará por ganado cuando estén TODOS los orcos de ese
 * nivel muertos (esto es, con vida menor o igual a 0).
 * Devolverá:
 * >  0 si el estado es jugando.
 * >  1 si el estado es ganado.
 */
int estado_nivel(nivel_t nivel);

```

```c
/*
 * Agregará un defensor en el nivel recibido como parametro.
 * Devolverá:
 * >  0 si pudo agregar el defensor correctamente.
 * > -1 si no pudo (la coordenada es parte del camino de ese nivel, existe otro defensor, etc.)
 */
int agregar_defensor(nivel_t* nivel, torres_t* torres, coordenada_t posicion, char tipo);

/*
 * Jugará un turno y dejará el juego en el estado correspondiente.
 * Harán su jugada enanos, elfos y orcos en ese orden.
 */
void jugar_turno(juego_t* juego);

/*
 * Mostrará el mapa dependiendo del nivel en que se encuentra el jugador.
 */
void mostrar_juego(juego_t juego);

#endif /* __DEFENDIENDO_TORRES_H__ */
```

4.2. Convenciones

Se deberá utilizar la siguiente convención:

- **Elfos**: L.
- **Enanos**: G.
- **Orcos**: O.
- **Torre**: T.
- **Entrada**: E.

4.3. Biblioteca animos.h

Se debe crear una biblioteca con el trabajo realizado en la Parte 1, ésta biblioteca solo tendrá un procedimiento con la siguiente firma:

```c
void animos(int* viento, int* humedad, char* animo_legolas, char* animo_gimli);
```

5. Resultado Esperado

Se espera que se creen las funciones y procedimientos para que el juego se desarrolle con fluidez.

Muchas de las funcionalidades quedan a criterio del alumno, solo se pide que se respeten las estructuras y especificaciones brindadas.

El trabajo creado debe:

- Interactuar con el usuario.
- Mostrarle al jugador la información del juego a cada momento.
- Informarle al jugador correctamente cualquier dato que haya sido ingresado incorrectamente.
- Informarle al jugador si ganó o perdió.
- Cumplir con las buenas prácticas de programación.
- Mantener las estructuras propuestas actualizadas a cada momento.

6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado **juego.c**, y la biblioteca de funciones para jugarlo **defendiendo_torres.c** y **defendiendo_torres.h**, y debe poder ser compilado sin errores con el comando:

```
gcc juego.c defendiendo_torres.c animos.c utiles.o -o juego -std=c99 -Wall -Wconversion -Werror -lm
```

animos.c y **animos.h** corresponden a la biblioteca creada con la parte 1 del trabajo para obtención de los ánimos de Legolas y Gimli.

utiles.o es un archivo compilado realizado por la cátedra, que pondrá a su disposición 2 funciones que pueden ser, justamente, útiles y su funcionamiento se explica en el anexo.

Por último debe ser entregado en la plataforma de corrección de trabajos prácticos Kwyjibo en la cual deberá aparecer con la etiqueta successful.

Para la entrega en Kwyjibo, recuerde que deberá subir un archivo zip conteniendo únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

7. Anexo

7.1. Distancia Manhattan

Para obtener la distancia entre 2 puntos mediante este método, se debe conocer a priori las coordenadas de dichos puntos.

Luego, la distancia entre ellos es la suma de los valores absolutos de las diferencias de las coordenadas. Se ve claramente en los siguientes ejemplos:

- La distancia entre los puntos (0,0) y (1,1) es 2 ya que: |0 - 1| + |0 - 1| = 1 + 1 = 2
- La distancia entre los puntos (10,5) y (2,12) es 15 ya que: |10 - 2| + |5 - 12| = 8 + 7 = 15
- La distancia entre los puntos (7,8) y (9,8) es 2 ya que: |7 - 9| + |8 - 8| = 2 + 0 = 2

7.2. Obtención de Números Aleatorios

Para obtener números aleatorios debe utilizarse la función **rand()**, la cual está disponible en la biblioteca stdlib.h.

Esta función devuelve números pseudo-aleatorios, esto quiere decir que, cuando uno ejecuta nuevamente el programa, los números, aunque aleatorios, son los mismo.

Para resolver este problema debe inicializarse una semilla, cuya función es determinar desde donde empezarán a calcularse los números aleatorios.

Los números arrojados por **rand()** son enteros sin signo, generalmente queremos que estén acotados a un rango (queremos números aleatorios entre tal y tal). Para ésto, podemos obtener el resto de la división de **rand()** por el valor máximo del rango que necesitamos.

Aquí dejamos un breve ejemplo de como obtener números aleatorios entre 10 y 30.

```c
#include <stdio.h>
#include <stdlib.h> // Para usar rand
#include <time.h>   // Para obtener una semilla desde el reloj

int main(){
        srand ((unsigned)time(NULL));
        int numero = rand() % 20 + 10; // la amplitud del rango es 20 y el valor mínimo es 10.
        printf("El valor aleatorio es: %i\n", numero);

        return 0;
}
```

7.3. Limpiar la Pantalla durante la Ejecución de un Programa

Muchas veces nos gustaría que nuestro programa pueda verse siempre en la pantalla sin ver texto anterior.

Para ésto, podemos utilizar la llamada al sistema **clear**, de esta manera, limpiaremos todo lo que hay en nuestra terminal hasta el momento y podremos dibujar la información actualizada.

Y se utiliza de la siguiente manera:

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

int main(){
        printf("Escribimos algo\n");
        printf("que debería\n");
        printf("desaparecer...\n");

        system("clear"); // Limpiamos la pantalla

        printf("Solo deberiamos ver esto...\n");
        return 0;
}
```

7.4. utiles.o

En esta oportunidad la cátedra pondrá a disposición un biblioteca con 2 funciones, una para obtener un camino aleatorio entre 2 puntos y otra para detener el tiempo por un tiempo.

Es pertinente aclarar que se envían 2 archivos **utiles.o**, uno para arquitecturas de 32 bits y otra para arquitecturas de 64 bits, use la que corresponda a su sistema operativo.

El .h de la biblioteca se muestra a continuación con las firmas propuestas.

```c
#ifndef UTILES_H
#define UTILES_H

#define MAX_LONGITUD_CAMINO 200

typedef struct coordenada{
        int fil;
        int col;
} coordenada_t;

/*
 * Generará un camino aleatorio desde la entrada hasta la torre.
 * Las coordenadas de entrada y de la torre deben ser válidas.
 */
void obtener_camino(coordenada_t camino[MAX_LONGITUD_CAMINO], int* tope_camino, coordenada_t entrada
    , coordenada_t torre);

/*
 * Detendrá el tiempo los segundos indicados como parámetro.
 * Ejemplos:
 * - Para detener medio segundo se deberá llamar detener_el_tiempo(0.5)
 * - Para detener dos segundos se deberá llamar detener_el_tiempo(2)
 */
void detener_el_tiempo(float segundos);

#endif
```

De más está decir que no es obligatorio el uso de éstas funciones, pero la generación de un camino aleatorio resuelve un problema que puede llevarles demasiado tiempo y detener el tiempo les permite acelerar o detener la ejecución del juego, en valores mas flexibles, ya que existe una función llamada **sleep**, pero su mínimo valor es 1 segundo.