

A capturar la bandera !!

Aclaraciones

- Para aprobar la totalidad del TP es necesario tener aprobado cada uno de sus módulos.
- Fecha de entrega: Domingo 06 de Noviembre de 2022



Objetivo

El objetivo del TP es programar el famoso y archiconocido juego “Capturar la bandera”.

En dicho juego participan dos Oponentes o **Equipos**. Los llamaremos, para simplificar, equipo *Rojo* y equipo *Azul*. Cada equipo tendrá a su vez una cierta cantidad de **Jugadores**.

Desarrollo del juego

El juego consiste en llegar con algún jugador de su color a la bandera contraria.



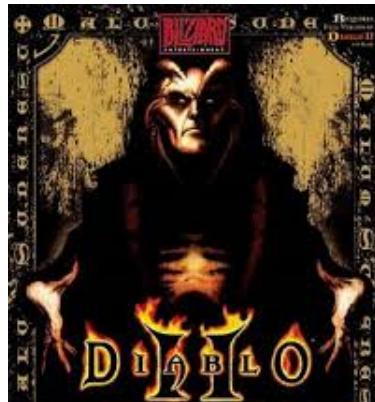
Reglas de juego

Nos han pedido programar la **Inteligencia** de los jugadores de un equipo.

Las reglas del juego son las siguientes:

- Comienza moviendo el equipo **Rojo**
- Los equipos se mueven por turno, primero el **Rojo**, luego **Azul** y así sucesivamente hasta que haya un ganador.
- Un jugador no puede pasar por sobre otro jugador (contrario o propio).
- Nunca un jugador puede salir de la grilla.
- Los jugadores sólo pueden moverse hacia arriba, abajo, izquierda o derecha.
- El equipo que logre que algún jugador suyo llegue antes a la bandera del contrario gana.

Para lograr jugar se contará con un **Game Master** o **Belcebú** que será el encargado de dirigir el juego.



Software

Para realizar el juego utilizaremos `std::thread` o `pthreads` (en general `std::thread` termina siendo implementada sobre `pthreads`, como sea de su preferencia.)

Para esto ustedes programarán el código de los threads de sus jugadores y algunos otros según se detalla.

Esquema del sistema

Se cuenta con la clase `config` que es la encargada de tomar los datos. Los mismos serán ingresadas vía el archivo `./config/config_parameters.csv`, y cuya documentación se encuentra en el archivo `./config/documentacion.txt`

En el mismo se definirá:

- `ValorX` y `ValorY` serán la cantidad de columnas y filas del tablero respectivamente.
- `CantidadJugadores` la cantidad de jugadores por equipo.
- `BanderaRojaX` y `BanderaRojaY` la posición de la bandera del equipo rojo.
- `BanderaAzulX` y `BanderaAzulY` la posición de la bandera del equipo azul.
- `coordinadasInicialesRojo` lista de X e Y de posiciones de iniciales de jugadores rojos.
- `coordinadasInicialesAzul` lista de X e Y de posiciones de iniciales de jugadores azules.

Esto es utilizado para ser inicializado en `main.cpp` el gameMaster y los equipos.

A su vez en dicho archivo podrá encontrar dos constantes para probar sus estrategias, definiendo la estrategia a utilizar `const estrategia strat = SECUENCIAL;` y el quantum `const int quantum = 10;` que podrán modificar para correr sus estrategias.

Un Equipo (`equipo.cpp` y `equipo.h`) tomará los siguientes datos:

- un puntero al objeto `belcebu`.
- el color de su equipo.
- la estrategia a utilizar.
- la cantidad de jugadores del equipo.
- un entero indicando cuántas movidas puede hacer el equipo por turno o **Quantum** (si no es utilizada en la estrategia, este dato debe ser pasado aunque podría no ser utilizado)
- una tupla de enteros contenido el x e y de ese jugador.

Una vez creados los equipos comenzará el juego mediante el método `comenzar` del Equipo.

Para moverse, un jugador de un equipo deberá llamar al método `belcebu->mover_jugador(dir, nro_jugador);` de `belcebu`, que mueve al jugador 1 sola casilla. De querer moverlo más de 1, deberá llamarlo varias veces. La dirección puede ser alguna `direccion`. El mismo devuelve el nro de ronda o 0 si el equipo ganó.—

Cuando un equipo termine de realizar sus movidas, el último jugador del del mismo en moverse (o el equipo), deberá llamar al método `belcebu->termino_ronda(equipo);` de `belcebu`, indicando el equipo al que pertenece dicho jugador.

Se podrán generar todas las variables globales que necesite para realizar los objetivos, pero las mismas nunca podrán ser compartidas con las del otro equipo.

El juego termina cuando alguno de los dos equipos gana.



Estrategias

Las estrategias, como se mencionó previamente, son únicamente para determinar qué jugador mover. El programador tendrá potestad absoluta para decidir hacia donde moverlo.

A continuación detallamos las estrategias posibles:

1. *Secuencial*. Sin determinar orden, mueve cada jugador un paso hacia algún objetivo. Al mover todos los jugadores de un equipo, le tocará al otro. Si un jugador está rodeado y no puede moverse, pierde su turno.
2. *Round Robin*. Moverá *Quantum* pasos, cada equipo. De no alcanzar la cantidad de jugadores, se moverá 1 vez cada jugador. De ser $Quantum > N$, con N la cantidad de jugadores, volverán a jugar los jugadores hasta terminar el quantum. Deberán jugar en órden según su número de jugador `nro_jugador`.
3. *Shortest distance first* (menor distancia a la bandera primero). Se deberá mover el jugador que está más cerca de la bandera. Un jugador de cada equipo por vez.
4. *Estrategia de ustedes*. Estrategia a definir por ustedes. La misma deberá utilizar el *Quantum* para definir cuántos pasos moverse en cada ronda (por cada equipo).

Ejercicios

Para realizar el tp, comenzaremos con una implementación bottom up, es decir, a partir de las clases más simples hasta las más complejas. El sistema debe ser realizado para que funcione adecuadamente según las especificaciones anteriormente mencionadas.

1. Hacer las implementaciones de belcebú (`gameMaster`).
 - Constructor `gameMaster`. Deberán completarlo para poder sincronizar adecuadamente.
 - `mover_jugador(direccion dir, int nro_jugador)`. Mueve al jugador definido hacia una dirección. Dicho método deberá ser llamado por el jugador indicado en el parámetro.
 - `termino_ronda(color equipo)`. Deberá ser llamado por un jugador del equipo en turno al ser el último jugador juega en su equipo en esa ronda.
2. Desarrollar los métodos de los equipos (`equipo.h` y `equipo.cpp`)
 - `Equipo::Equipo(...)`. El constructor de los equipos. Tal vez necesiten realizar alguna inicialización de sincronismo.
 - `void Equipo::comenzar()`. Define quién comienza y lanza los threads de los jugadores.
 - `void Equipo::jugador(int nro_jugador)`. Es el método que es llamado en un thread. Define a los jugadores. Las estrategias deben ser llamadas desde allí.
3. Realizar las estrategias mencionadas.
4. Defina el método `coordenadas Equipo::buscar_bandera_contraria()` de la clase `equipo` donde los jugadores se repartan el tablero y puedan buscar, de manera paralela/simultánea, la bandera contraria. En particular, será de interés medir los tiempos de búsqueda de la bandera contraria utilizando o no el paralelismo mencionado anteriormente. Para hacerlo pueden utilizar la función `clock_gettime` (con la opción `CLOCK REALTIME`) de la biblioteca `time.h`, que se puede linkear utilizando `-lrt`.

Nota: No podrá serializar la ejecución de los jugadores cuando no lo pida el enunciado ni bloquear innecesariamente a los jugadores. Podrá generar nuevos atributos, métodos o clases que crea necesarios

Entregables



- El sistema deberá utilizar C++ y generar un ejecutable que realice lo mencionado anteriormente.
- Casos de test mostrando el funcionamiento correcto para cada operación realizada.
- Deberá realizar un informe explicando lo realizado y cualquier comentario que crea relevante para que los docentes puedan entender lo realizado.
- Deberá enviar un mail con un adjunto .zip (o .tar.gz) a la cuenta so-doc con el asunto TP1GX-SO-2022 donde X será su número de grupo. En el cuerpo del email enviar los integrantes del grupo y su nro. de libreta.
- El zip deberá contener el .pdf del informe y una carpeta src con los fuentes. Las mismas deberán compilar con el comando “make”.