1 Seminarios C y C++

Los requerimientos de cada ejercicio del seminario serán expuestos desde el punto de vista práctico y teórico; es decir, para su exposición, cada equipo se basará en el caso práctico en cuestión para introducir y explicar el elemento teórico requerido. La exposición no es una mera enunciación de código. Preguntas como: *¿Por qué?*, *¿Basándose en qué?*, *¿Cómo se logra esto en el lenguaje* `X`*?* entre otras, deben hacerse.

Todos los miembros del equipo deben participar en la solución del ejercicio y estar preparados para exponer todo el trabajo. **La persona a exponer** se decide el día de la exposición. Quién no esté presente en la exposición de su equipo tiene `0` en la evaluación. (Note que estas notas se promedian y hay distinción entre 0 y 2).

1.1 Seminario 3 (C++11 y C++14)

Diseñe una jerarquía de clases en C++ que permita modelar las unidades y personajes del universo de **Starcraft**. Se ofrece a continuación una descripción de este mundo, junto con preguntas y elementos del lenguaje `C++` que deben ser utilizados en la modelación del mismo.

1. En este video-juegos existen 3 razas principales: **Terran**, **Protoss** y **Zerg**. Todas las unidades de cualquier raza tienen 3 valores enteros:

   - *la vida*
   - *índice de ataque*
   - *índice de defensa*.

   Los Protoss además poseen dos valores adicionales:

   - Un valor que representa su escudo de protección.
   - Un valor que representa cuánto escudo se regenera en un segundo.

   Los Zerg por otro lado tienen:

   - Un valor extra que representa cuánta vida regeneran en un segundo.

   a. Modele estos conceptos utilizando como clase base la mostrada en el código al final del ejercicio. Note que los constructores de las clases que no se deban poder instanciar deben ser *protected*.
   b. ¿Cuál es el tipo de herencia por defecto de `C++` (`public`, `protected` o `private`)?
   c. ¿Cómo se representa en memoria la herencia en `C++`?

```
class Unit{
    int _life, _attack, _defense;
    std::string _name;

protected:
    Unit(int life, int attack, int defense, std::string&& name)
        :_life(life), _attack(attack), _defense(defense), _name(name){}

public:
    void print()
    {
        std::cout << _name << std::endl;
    }
};
```

2. Además de estas 3 razas, existen 2 adicionales que son mezclas de las anteriores:

   - *Terran Infestados* (**Terran** - **Zerg**)
   - *Los Híbridos* (**Protoss** - **Zerg**)

   a. Modele esto utilizando herencia múltiple y *virtual*.
   b. ¿Qué problemas trae la herencia múltiple con respecto a la representación en memoria de la herencia?

3. Algunas unidades del juego poseen habilidades o poderes (hasta un máximo de 3). Este tipo de unidad también tienen 2 valores adicionales:

   - Un entero que representa la energía que poseen para lanzar sus poderes.
   - Un entero que representa la cantidad de energía que recuperan cada segundo.

   Existen 3 tipos de poderes:

   - Los poderes generales, los cuáles pueden ser poseídos por cualquier unidad de cualquier raza.
   - Los poderes específicos para cada raza, los cuáles pueden tener las unidades de cada raza respectivamente.
   - Los poderes especiales, que también están restringidos a cada raza, pero sólo pueden ser vinculados a las unidades especiales o héroes de cada raza.

Para modelar estos requisitos:

1. Utilice la siguiente clase base para la jerarquía de los poderes:

```
class Power
{
public:
    virtual ~Power(){}
    virtual void Cast(ManaUnit manaUnit) = 0;
};
```

2. Utilice la siguiente clase para describir a las unidades que tienen energía (*mana*) para canalizar poderes.

```
class Mana_Unit
{
    int _mana, _genManaPerSecond;

protected:
    ManaUnit(int mana, int genManaPerSecond)
    : _mana(mana), _genManaPerSecond(genManaPerSecond){}
};
```

3. A partir de estas clases cree una que represente las unidades con poderes. Esta clase debe ser genérica en 3 parámetros que representan los poderes que se les puede asignar. Se deben restringir los parámetros genéricos para que sólo acepten clases **concretas** que hereden de *Power*. Sólo el primer parámetro genérico debe ser obligatorio, los demás pueden no asignarse.
   a. Genericidad con `templates`.
   b. Valores por defecto a los `templates`.
   c. Restricciones sobre los `templates`.
4. Cómo se mencionó anteriormente, cada raza tiene unidades únicas o héroes, los cuales pueden poseer poderes especiales y de los cuales sólo puede existir una instancia.
   a. Explicar el patrón *Singleton*.
5. Implemente algunos ejemplos de poderes, unidades y héroes.
   a. Explicar la especialización de `templates`.