ALGORITMOS (CC215)
Ciclo 2024-2
Examen Parcial

Profesores: TODOS
Duración: 170 minutos
Secciones: TODAS

DECLARACIÓN.
El alumno debe declarar si ha utilizado herramientas de IA en la solución del examen parcial. Además, debe especificar en qué parte de su solución las ha aplicado. Si el alumno usa alguna herramienta de IA y no lo declara, el docente tiene la potestad de penalizar la calificación si detecta su uso.
Asimismo, el docente puede solicitar a al alumno que demuestre su conocimiento del código de programación enviado. Si algún estudiante no demuestra conocer su código, el docente tiene la potestad de penalizar la calificación.

INSTRUCCIONES:

- ✓ La solución debe subirse al aula virtual de Blackboard con la siguiente nomenclatura: EA1-CODIGO-ALUMNO.zip
- ✓ Se permite usar materiales desarrollados en clase que solo se encuentran en el aula virtual de la UPC, accediendo únicamente a: los ejercicios, código fuente compartido en clase, librerias personasles. No se puede acceder a código fuente correspondiente a actividades grupales como: trabajo parcial, final, GameJam, etc.
- ✓ Las herramientas a usar son: Visual Studio 2019 o superior, StarUML, LucidChart, paint.

Ejercicio 1

De acuerdo con el código que se encuentra líneas abajo, se solicita desarrollar lo siguiente:

```cpp
#include <iostream>
using namespace std;

class Animal {
public:
    virtual double calcularVelocidad(){}
    virtual double calcularFuerza() {}
};

class Perro : public Animal {
private:
    string raza;
    double tamano;
public:
    Perro(string r, double t) : raza(r), tamano(t) {}

    double calcularVelocidad() {
        return tamano * 2;
    }

    double calcularFuerza() {
        return tamano * 3;
    }
};

// Continúa el código con las demás clases...
```

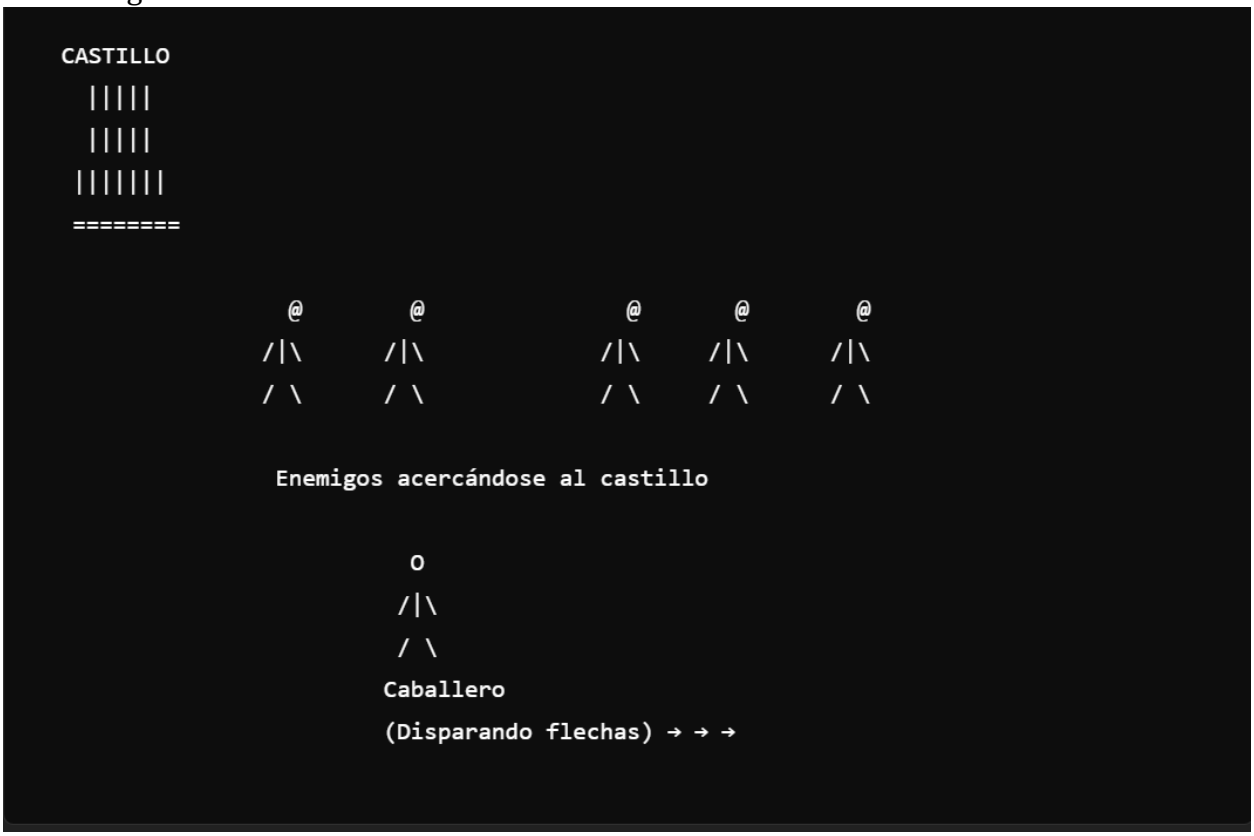
- Analice el siguiente código que contiene la clase base "Animal" para determinar el caso que represente Herencia.
- Agregue las clases "Conejo" y "Tortuga" (considerar que tienen atributos: velocidad y tamaño).
- Agregue el elemento .CPP que permita gestionar los 6 tipos de animales y realizar las siguientes operaciones:
- Cuando se escoja el Perro, se ingrese la raza y tamaño para calcular la velocidad y fuerza.
- Cuando se escoja el Gato, se ingrese el tamaño y peso para calcular la agilidad y velocidad.
- Cuando se escoja el Caballo, se ingrese la altura y peso para calcular la fuerza y velocidad.
- Cuando se escoja el Pájaro, se ingresen las dimensiones de las alas para calcular la velocidad de vuelo.

RÚBRICA DE CALIFICACIÓN

Ítems	Experto	Básico	Deficiente	Descripción
Herencia aplicada correctamente	**1.0**	**0.5**	**0**	Se espera que el uso de la herencia esté bien aplicado, respetando la jerarquía de clases y con un buen manejo de los atributos y métodos en las clases derivadas.
Clases pendientes agregadas	**2.0**	**1.0**	**0**	Implementa las clases Conejo, Tortuga, Gato, Caballo y Pájaro
Código funcional	**2.0**	**1.0**	**0**	El programa solicita la información correspondiente por cada tipo de animal

Ejercicio 2 Defensa del castillo

En este caso, debes desarrollar un mini juego llamado DEFENSA DEL CASTILLO en el cual los jugadores deben defender un castillo del ataque de diferentes tipos de enemigos. El objetivo es que el jugador controle un Caballero y derrote a los enemigos antes de que lleguen al castillo. El juego debe desarrollarse utilizando programación orientada a objetos y cumplir con las siguientes características:



Clase CABALLERO

- Representado por el siguiente arte ASCII:

```
 O
/|\
/ \
```

- Es de color azul.
- Se mueve en las cuatro direcciones por la pantalla utilizando las teclas direccionales.
- Tiene 3 vidas.
- Puede atacar a los enemigos cuando estos se encuentran a una casilla de distancia.
- Su objetivo es proteger el castillo eliminando a los enemigos antes de que lleguen

Clase ENEMIGO

- Representado por el siguiente arte ASCII:

```
 @
/|\
/ \
```

- Es de color rojo.
- Se mueve automáticamente en dirección hacia el castillo.
- Si un enemigo llega al castillo, el jugador pierde una vida.
- Hay un total de 5 enemigos que aparecerán aleatoriamente dentro del escenario y moverse hacia el castillo.

Clase CASTILLO

Representado por el siguiente arte ASCII:

```
 |||||
 |||||
|||||||
=====
```

- Es de color gris.
- Se encuentra en el centro de la pantalla, y los enemigos tratarán de llegar a él.
- El castillo no se mueve.
- Si todos los enemigos logran llegar al castillo, el jugador pierde la partida.

Clase ARCO

- Representado por el siguiente símbolo ASCII (→):

→

- Es el arma del caballero.
- Permite atacar a los enemigos que están a una casilla de distancia.
- Si chocan con el enemigo ambos desaparecen.
- El caballero puede disparar en las cuatro direcciones presionando la tecla P
- Las balas son ilimitadas

Funcionamiento del programa

El programa debe cumplir con los siguientes requisitos:

- El jugador controla al caballero con las teclas direccionales.
- Los enemigos se mueven automáticamente hacia el castillo.
- El caballero debe disparar flechas (→) hacia los enemigos para eliminarlos.
- Si un enemigo llega al castillo, el jugador pierde una vida.
- Si el jugador pierde todas las vidas, se mostrará el mensaje "DERROTA".
- Si el jugador elimina a todos los enemigos antes de que lleguen al castillo, se mostrará el mensaje "VICTORIA".

Ítems	Experto	Básico	Deficiente	Descripción
Relacion de Herencia	**2.0**	**1.0**	**0**	Se espera que el uso de la herencia esté bien aplicado, respetando la jerarquía de clases y con un buen manejo de los atributos y métodos en las clases derivadas.
Diagrama de clases	**2.0**	**1**	**0**	Debe presentar un diagrama de clases que muestre claramente la relación entre las clases y cómo se organizan en la jerarquía del sistema, siguiendo las convenciones de UML.
Clases pendientes agregadas	**2.0**	**1.5**	**0**	Las clases solicitadas (como "Enemigo" y "Castillo") deben estar correctamente implementadas con los atributos y métodos requeridos, siguiendo el principio de POO, polimorfismo, reutilización de código, entre otros conceptos enseñados en clase.
Código funcional	**2.0**	**1.0**	**0**	El código debe funcionar correctamente, sin errores de compilación o ejecución, y debe cubrir todas las funcionalidades pedidas (movimiento, colisiones, interacciones).
Movimiento y colisiones	**3.0**	**1.5**	**0**	El movimiento del caballero y los enemigos, así como las colisiones entre personajes, recursos y obstáculos, deben implementarse correctamente y de forma fluida.
Interacción del jugador (caballero y disparos)	**2.0**	**1.0**	**0**	La interacción del jugador con el caballero, incluyendo el control del personaje mediante teclas y el uso de disparos para eliminar enemigos, debe estar bien implementada.
Finalización del juego	**2.0**	**1.0**	**0**	El juego debe terminar correctamente, mostrando el mensaje adecuado de victoria o derrota según las condiciones del juego. El jugador debe saber cuándo ha ganado o perdido.