

Nombre: _____ Curso: _____	<u>Nota</u>
-----------------------------------	--------------------

Cosas permitidas y no permitidas:

- No se puede consultar Internet
- El código entregado tiene que compilar sin problemas, si no, no se corregirá el examen
- Cada error, falta de implementación o situación no controlada restará puntos dependiendo de la gravedad
- **A lo largo del examen deberás tomar varias decisiones que no están escritas en el examen.**



Este ejercicio consiste en hacer una simulación de la carrera de los autos locos.

Las normas de la simulación son:

- Hay 4 participantes (en la carrera original había más) y varios obstáculos
- En cada coche hay un piloto, y puede tener un copiloto.
- ***Es un sistema que funciona por turnos***, en cada turno actuarán todos los objetos de la carrera.
- El que llegue antes a la meta gana, pero puede haber algún empate.

Las clases a implementar son las siguientes:

IRace

- Es una interfaz
- Métodos
 - **AddObject(obj: RaceObject, position: real)** → Añade un objeto a la carrera
 - **void Init(distance: real)** → Es un método que inicia la carrera.
 - **void SimulateStep()** → Es un método que realizará SOLO un turno en la carrera.
 - **void VisitDrivers(lambda)** → Método que recibe una lambda y recorre todos los pilotos y copilotos.
 - **void VisitCars(lambda)** → Método que recibe una lambda y recorre todos los coches.
 - **void VisitObstacles(lambda)** → Método que recibe una lambda y recorre todos los obstáculos.
 - **void VisitObjects(lambda)** → Método que recibe una lambda y recorre todos los objetos.
 - **int GetObjectCount()** → Devuelve el número de Objetos de la carrera.
 - **GetObjectAt(index)** → Devuelve el objeto de la carrera que está en la posición index.

Race

- Es una clase que implementa la interfaz IRace

RaceObject

- Es una clase
- Tiene los siguientes atributos **privados** que no se pueden modificar de manera pública:
 - **name** → Tiene una property con un getter
 - **position** → En un real que dictamina a qué distancia se encuentra desde la posición de salida. Las clases hijas pueden modificar este valor, se debe poder acceder de manera pública.
- Tiene las siguientes propiedades
 - **bool IsAlive** → Es abstracto. Devuelve si el objeto está activo o no. Si un objeto no está activo, debe borrarse de la lista de objetos de la carrera.
- Tiene los siguientes métodos públicos
 - **ObjectType GetObjectType()** → Es abstracto. Devuelve el tipo de objeto.
 - **void Disable(int turns)** → Hace que el objeto no se pueda mover durante un número determinado de turnos. Esos turnos son acumulables.
 - **void Simulate(IRace)** → Función que ejecuta el comportamiento del objeto. Toma las decisiones que creas convenientes.



Obstacle

- Hereda de RaceObject

Rock

- Hereda de Obstacle
- Cada roca tiene una propiedad llamada peso que va de 10 a 30.
- En su turno la roca mirará los objetos que se encuentran a menos de 40 metros de ella e intentará hacer que se retrase. Para ver si la piedra tiene éxito, hay que generar un random. Cada objeto tiene un 10% + el peso de la piedra de probabilidad de retrasarse 25 metros.

Puddle (charco)

- Hereda de Obstacle
- En su turno mira qué objetos hay a menos de 20 metros de él, para cada objeto, hay un 20% de probabilidades de que lo desactive entre 0 y 3 turnos.

Bomb

- Hereda de Obstacle
- Cuando se pone una bomba en la carrera, explotará en un número determinado de turnos que se le pasará por el constructor.
- Cuando explote, todo vehículo que se encuentre a menos de 70 metros de ella, se desplazará una distancia que vá desde -50 metros a +50 metros.
- Cuando la bomba haya explotado, la función IsAlive, devolverá false, por lo que deberá borrarse de la carrera.

Car

- Hereda de RaceObject
- Tiene un atributo que sólo pueden ver sus clases hijas. Ese atributo se llama finetunning. Es un valor real aleatorio entre 1 y 3 que se le pasará por el constructor. Cada vez que un coche se mueva, su desplazamiento se ve incrementado en esta cantidad.

GlamourCar

- Hereda de Car
- Tiene una piloto
- Va a un ritmo constante y seguro de 20 metros por turno.

TroglodyteCar

- Hereda de Car
- Tiene 2 pilotos
- Va a un ritmo de 10 metros por turno.
- De vez en cuando (el 30% de las veces) decidirán golpearse la cabeza, cuando lo hagan, tienen un 40% de posibilidades de avanzar 20 metros extra, y un 20% de quedarse desactivados el turno siguiente.



WoodCar

- Hereda de Car
- Tiene 1 piloto y un copiloto que es un castor
- Va a 15 metros por turno.
- Cuando este coche está desactivado, ..., por ejemplo por que se ha metido en un charco, tiene un 60% de probabilidades de volver a activarse y seguir compitiendo.

PiereCar (Implementar esta clase se ponderará con más nota que los demás corredores)

- Hereda de Car
- Tiene 1 piloto y un perro copiloto (Patán)
- Va a 18 metros por turno.
- Tiene de 10 a 20 trampas que podrá colocar para que los coches se retrasen: De vez en cuando (el 50% de las veces) tiende una trampa, esto es, elegirá al coche que va justo detrás de él y tiene un 30% de probabilidades de desactivarlo un turno. Si falla, risitas (Patán) se baja del coche y se va a su casa riéndose.

Driver

- Es una clase
- Es la clase que servirá para manejar pilotos y copilotos
- **GetVelocityExtra()** → Es una función abstracta. En cada turno, cada coche llamará a esta función de cada piloto y copiloto que tenga. El valor devuelto, se sumará a la distancia recorrida ese turno.

Animal

- Hereda de Driver
- **GetVelocityExtra()** devuelve entre -1.0 y 3.0

Human

- Hereda de Driver
- **GetVelocityExtra()** devuelve 0.0

INDISPENSABLE: Has de desarrollar (y entregar) un main en el cual:

1. Ha de insertar 4 participantes, uno de cada clase y ponerlos en la línea de salida
2. Ha de insertar un obstáculo de cada tipo en alguna posición. Tanto la posición como las características de los obstáculos serán aleatorios entre sus valores permitidos.
3. Iniciar la carrera.
4. Simular la carrera hasta que haya un ganador/a (o varios)
5. Imprimir por pantalla el ganador

