# Parcial Diseño de Aplicaciones 1 – MATUTINO – Julio 2018

**Duración:** **3 horas**
**Con material:** **NO**
**Puntaje mínimo / máximo:** **20 / 40 puntos**
**Importante:**

- Escribir con letra clara.
- Todos los diagramas deben ser completos, prolijos y claros, incluyendo todos aquellos adornos UML, asegurando la correctitud y no ambigüedad.
- La ausencia de cualquiera de los puntos anteriores puede implicar la quita de puntos.

## Pregunta 1: Principios, Patrones, GRASP (20 puntos)

Se desea diseñar un videojuego. La solución actual pensada es modelar el juego como un conjunto de objetos de juego.

- Cuando el juego inicia, se configura la librería de gráficos, se cargan los objetos del juego (actores y partículas) y se asigna el nivel de dificultad.

- Luego, cada 15 milisegundos el juego se actualiza. Esto consiste en actualizar los objetos del juego: si el mismo está listo (ShouldUpdate()), se actualiza su física llamando a UpdatePhysics() y luego su lógica llamando a UpdateLogic().

- Cada objeto define los pasos de actualización de física y lógica.

Se tiene entonces el **siguiente** código:

```
public class Game
{
    private const float UPDATE_TIME = 15;
    private bool isGameRunning;
    private int difficulty;
    private Render lowLevelRender;
    private IEnumerable<Actor> actors;
    private IEnumerable<Particle> particles;

    public void Play()
    {
        SetUpGraphicsRenderer();
        LoadGameObjects();
        difficulty = 2;

        while (isGameRunning)
        {
            if (Time.deltaTime > UPDATE_TIME)
            {
                foreach (var actor in actors)
                {
                    if (actor.ShouldUpdate())
                    {
                        actor.UpdatePhysics();
                        actor.UpdateLogic();
                    }
                }
                foreach (var particle in particles)
                {
                    if (particle.ShouldUpdate())
                    {
                        particle.UpdatePhysics();
                        particle.UpdateLogic();
                    }
                }
            }
        }
        Render();
    }

    private void SetUpGraphicsRenderer()
    {
        SystemGraphics graphics = new SystemGraphics();
        if (graphics.Drivers.Render.ApiName == "OpenGL_4.x")
            lowLevelRender = new OpenGLRender();
        else if (graphics.Drivers.Render.ApiName == "DX12")
            lowLevelRender = new DirectX12Render();
    }

    private void Render()
    {
        //[LETRA: Código de renderizado]
        //...
```

**Se pide:**

a) **(4 puntos)** Modele la solución actual con un **diagrama de clases UML**. El diagrama debe ser lo más completo y prolijo posible utilizando correctamente los **adornos** UML.

b) **(5 puntos)** Discuta brevemente si la clase Game cumple con los siguientes principios de diseño: **SRP, OCP y DIP**. Brinde un ejemplo de porqué cumple o no con cada principio.

c) **(3 puntos)** Realice un diagrama de secuencia del método Play() de la clase Game.

d) **(5 puntos)** Indique qué patrón de diseño se podría aplicar para los objetos del juego y el loop de actualización de los mismos.
   i) Justifique por qué eligió ese patrón.
   ii) Enuncie la intención de dicho patrón.
   iii) Diagrame su solución utilizando dicho patrón.

e) **(3 puntos)** La clase **Time** pertenece a una librería que desarrolló otro equipo. Ahora encontró una nueva y más rápida librería que ofrece una clase similar llamada **FastTime.** Dentro de las clases Game, Actor, Particle y otras, usted está utilizando la clase **Time.** Por ahora utiliza **Time.deltaTime** pero no está seguro si en un futuro tenga que cambiarlo a **FastTime.deltaTime**.

```
public class Time
{
    public static float deltaTime;
    public static float fixedDeltaTime;
    public static float smoothedDeltaTime;
    ....
}
public class FastTime {
    public static float deltaTime;
    public static float fixedDeltaTime;
    public static float smoothedDeltaTime;
    ....
}
```

**Se pide entonces:** ¿Cómo minimizaría el riesgo a futuro de reemplazar una por otra? ¿Hay algún **GRASP** que lo ayude en esta tarea? Justifique.

**Pregunta 2: Clean Code (7 puntos)**

a) **(4 puntos)** Según clean code:
   i) ¿Cuántos niveles de abstracción tiene el método **Play()**? Justifique
   ii) ¿Dicha cantidad de niveles de abstracción es recomendable? Justifique
   iii) Brinde un ejemplo que viole Clean Code en el método **Play()** y otro en el método **SetUpGraphicsRenderer().**

b) **(3 puntos)** Según clean code:
   i) ¿Cuántas responsabilidades debería tener un método?
   ii) Basado en su respuesta del punto i) y en lo que clean code sugiere para el manejo de excepciones, el **siguiente** ejemplo ¿es correcto o incorrecto? Justifique y brinde un ejemplo de cómo lo solucionaría.

```
public void UpdatePhysics()
{
    try
    {
        position.Set(speed * Time.deltaTime);
        rotation.Set(angularVelocity * Time.deltaTime);
    }catch(PhysicsEngineException e)
    {
        DropPhysicsUpdate(e);
    }
}
```

**Pregunta 3: TDD (8 puntos)**

a) **(4 puntos)** Explique cuál es el **proceso de TDD**. ¿Es posible y/o aceptable que algunos miembros de un equipo lo apliquen y otros no? ¿Qué impacto tendría en el código?
b) **(4 puntos) Time.**deltaTime de la pregunta 1 devuelve cuánto tiempo pasó entre un llamado y el siguiente a dicha propiedad en un bucle de actualización del Juego. ¿Una clase que dependa de Time.deltaTime es fácil de probar? ¿Existen inconvenientes? ¿Cuáles? Explique según **FIRST**.

**Pregunta 4: Tecnología (5 puntos)**

a) **(3 puntos)** Describa todos los pasos involucrados en la compilación de código c# y el proceso de ejecución de una aplicación .NET.
b) **(2 punto)** Cuando utilizamos Entity Framework, explique qué pasa al agregar a un DbSet de un Contexto una entidad que se relaciona con otras.