ChatGPT
Operator
image generator
Explore GPTs

Today
Juego carreras programación parale
PC questions
Ramdom

Previous 7 Days
Grammar Correction Assistance
Translate

Previous 30 Days
Programacion

2024
English
Incrementar ahorros y trabajo
Learn English Better!
Be better

crea un juego de carreras utilizando programacion paralelsa siguiendo los siguiente Explicación

En este ejercicio final, usted debe aplicar todos los conocimientos adquiridos hasta el momento para crear un juego simple que utilice programación paralela. El juego puede ser cualquier cosa. Para esto, puede utilizar una inteligencia artificial de su preferencia y ejecutar el siguiente prompt:

"Crea un ejercicio práctico en C# que permita a los estudiantes avanzados implementar un sistema de simulación, incorporando conceptos relacionados con tareas asíncronas como Task.Run, Task.Factory.StartNew, Task(...), Task.Factory.ContinueWhenAny, TaskCreationOptions.AttachedToParent, ContinueWith, TaskContinuationOptions.OnlyOnRanToCompletion, TaskContinuationOptions.OnlyOnCanceled, Task.Delay y Task.WhenAny." asegurate de agregar los conseptos relacionados con la programacion paralea que explique al final de la explicacion, crea el juego de almenos 8 corredores con nombres personalizados, dichos nombres deben ser un juego de palabras en español, al finalizar la carrera debe mostrar los nombres de los corredores con su respectiva posicion y tiempo en realizar la carrera, los tiempos deben asemejarse a los tiempos de una carrera real ejemplo: 1st rayo mcquenn tiempo: 02 minutos : 35 segundos : 223 milesismas, y asi sucesivamente con cada corredor

Aquí tienes un ejemplo de un juego de carreras en C# que utiliza programación paralela para simular la competencia entre 8 corredores. Utiliza varios conceptos de programación paralela, como `Task.Run`, `Task.Delay`, y las continuaciones `ContinueWith` para manejar el flujo asíncrono.

csharp Copy Edit

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Threading;

class Program
{
    static async Task Main(string[] args)
    {
        var corredores = new List<Corredor>
        {
            new Corredor("Correos Rápidos"),
            new Corredor("Velocidad Divina"),
            new Corredor("Corre Camino"),
            new Corredor("Relámpago Fulminante"),
            new Corredor("El Impulso"),
            new Corredor("Rayo McQueen"),
            new Corredor("Viento en Popa"),
            new Corredor("Fuga Imparable")
        };

        var tasks = corredores.Select(corredor => SimularCarrera(corredor)).ToArray();

        // Esperar a que todas las tareas terminen
        await Task.WhenAll(tasks);

        // Mostrar los resultados
        var clasificados = corredores.OrderBy(c => c.Tiempo.TotalMilliseconds).ToList();
        Console.WriteLine("\nResultados de la Carrera:");
        for (int i = 0; i < clasificados.Count; i++)
        {
            var puesto = i + 1;
            var tiempo = clasificados[i].Tiempo;
            Console.WriteLine($"{puesto}° {clasificados[i].Nombre}: {tiempo.Minutes:D2}:{t
        }
    }

    static async Task SimularCarrera(Corredor corredor)
    {
        // Iniciar la carrera con un tiempo aleatorio
        var tiempoTotal = new Random().Next(120, 300) * 1000; // Tiempo entre 2 y 5 minuto

        var tareaCarrera = Task.Run(async () =>
        {
            // Simular el progreso de la carrera
            for (int i = 0; i < tiempoTotal; i += 1000)
            {
                // Simulando cada segundo de carrera
                await Task.Delay(1000);
            }

            corredor.Tiempo = TimeSpan.FromMilliseconds(tiempoTotal);
        });

        // Continuación que se ejecuta una vez que la tarea se completa
        tareaCarrera.ContinueWith(t =>
        {
            if (t.Status == TaskStatus.RanToCompletion)
            {
                Console.WriteLine($"{corredor.Nombre} ha terminado la carrera!");
            }
            else if (t.Status == TaskStatus.Canceled)
            {
                Console.WriteLine($"{corredor.Nombre} no terminó la carrera.");
```

```
        });

        // Esperamos a que la tarea termine
        await tareaCarrera;
    }
}

class Corredor
{
    public string Nombre { get; set; }
    public TimeSpan Tiempo { get; set; }

    public Corredor(string nombre)
    {
        Nombre = nombre;
    }
}
```

Explicación de conceptos utilizados:

1. `Task.Run`: Se utiliza para ejecutar la simulación de cada corredor en un hilo paralelo. Cada corredor "correrá" de manera independiente en su propia tarea.
2. `Task.Delay`: Se utiliza para simular la duración de la carrera. En el código, cada "segundo" de la carrera es simulado mediante `Task.Delay(1000)`, lo que retrasa la ejecución en 1 segundo.
3. `Task.WhenAll`: Permite esperar a que todas las tareas (carreras de todos los corredores) se completen antes de mostrar los resultados.
4. `ContinueWith`: Se usa para realizar una acción después de que una tarea se completa. Aquí, se imprime un mensaje cuando cada corredor termina la carrera.
5. `TimeSpan`: Se usa para almacenar el tiempo total que tarda cada corredor en completar la carrera.
6. **Tareas asíncronas y paralelismo**: La carrera de cada corredor se ejecuta en un hilo independiente (tarea paralela), lo que permite simular la carrera de manera asíncrona sin que un corredor afecte a otro.

Resultado esperado:

Cuando ejecutes el programa, simulará la carrera con los 8 corredores, generando tiempos aleatorios entre 2 y 5 minutos. Al finalizar, mostrará los corredores ordenados según su tiempo, similar a una competencia real:

yaml Copy Edit

```
Resultados de la Carrera:
1º Rayo McQueen: 02:35:223
2º Velocidad Divina: 02:42:001
3º Relámpago Fulminante: 03:02:123
4º El Impulso: 03:10:211
5º Fuga Imparable: 03:15:110
6º Correos Rápidos: 03:22:309
7º Viento en Popa: 04:00:100
8º Corre Camino: 04:30:500
```

Este código te proporciona un ejercicio práctico en programación paralela con simulación de carreras, utilizando técnicas avanzadas de manejo de tareas asíncronas.