Clase 19

Martes, 17 Octubre 2023

Indice

Funciones complejas de contenedores

- Es util borrar la cache cada vez que cambiamos los paquetes nugget. Se encuentra en Herramientas / Opciones / Administrar Paquetes Nugget / Borrar todas las caches
- El mejor lugar para cargar datos y variables de una aplicación gráfica es 'OnLoad()'
- Las funciones 'Random' que vamos a utilizar son dos:

```
'
public static int GetRandomInt(int min, int max)
{
    return random.Next(min,max);
}

public static double GetRandomDouble()
{
    return random.NextDouble();
}

public static double GetRandomDouble(int min, int max)
{
    return random.NextDouble() * (max + min) + min;
}
'
```

Clase 20

Jueves, 19 Octubre 2023

Indice

Interfaz gráfica

- Es posible crear objetos de forma dinámica. Para ello utilizamos un bucle 'for' para crear un numero 'i' de personajes

```
'
for(int i = 0; i<countEnemies; i++)
{
    Character character = new Character();
    list.Add(character);
}
'
```

- Estudio del color. RGB frente a CMYK.
  RGB es habitual utilizarlo cuando pintamos sobre fondos negros, pantallas.
  CMYK es habitual utilizarlo cuando pintamos sobre fondos blancos, papel. Lo normal es necesitar imprimir el color negro cuando interactuamos sobre pantallas blancas.
- Es mucho mas util para un programador pensar en tantos por uno.
- Para pintar todos los elementos de nuestra pantalla, utilizamos las listas.
- Existen dos formas de utilizar los parametros de las funciones:
  ' r = 1.0; g = 1.0; b = 1.0; Draw(r,g,b); '
  En el momento de dibujar recogemos las variables
  ' Draw(r = 1.0, g = 1.0, b = 1.0); '
  Primero asigna todos los valores, y después realiza la función Draw(); Es menos aconsejable.
- El **Draw()** no cambia valores ni posiciones. El **Draw()** unicamente dibuja. Tenemos el **Animate()** para realizar cambios en el dibujo.
- Si tenemos valores que son muy repetitivos y largos de escribir, lo mas conveniente es utilizar variables donde almacenar estos valores
  ' player[i] // Mejor en una variable Character pj; pj = player[i]; '
- Estudiamos la filosofía de la programación orientada a objetos.
  Vamos a cambiar el funcionamiento de la función **OnDraw()**.

```
'
public class World
{
    public void OnDraw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(r,g,b,a);
            canvas.DrawRectangle(x,y,w,h);
        }
    }
}
'


Por este otro, mejor orientado a los objetos.


'
public class World
{
    public void OnDraw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            list[i].Draw(canvas);
        }
    }
}
```

```
public class Character
{
    public void Draw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(r,g,b,a);
            canvas.DrawRectangle(x,y,w,h);
        }
    }
}
'
```

Aquí se ordena al elemento 'i' de la lista que tiene la clase 'Mundo', que se pinte. Esta orden invoca a la función de objeto de la clase Character, donde se establece como debe pintarse este personaje.
Hemos separado el funcionamiento de Draw(), de manera que ahora desde la clase superior, llamamos a la funcion del objeto instancia para que se pinte.
Es importante pasarle el 'Icanvas canvas' para que pueda utilizar la función del SDK.

- Siempre que utilizamos metodos de objeto y llamadas de ese objeto, se añade implicitamente una orden 'this.', que es invisible para el usuario gracias al runtime, y que apunta hacia el objeto que se encuentra en ese mismo momento en ejecución.

```
'
public class Character
{
    public void Draw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(this.r,this.g,this.b,this.a);
            canvas.DrawRectangle(this.x,this.y,this.w,this.h);
        }
    }
'
```

- A partir de ahora vamos a separar las clases según su jerarquía en el programa. Cada clase crea un objeto de una clase mas específica, de modo que el funcionamiento del programa sea, a la vez eficiente y sencillo de entender.
- Debemos crear las siguientes clases en nuestro programa: Program, MyGame, World, Character y Utils
- Las Listas, realmente son arrays con Metodos diseñados por los programadores de C# para facilitar su uso.
- El Program perfecto es el siguiente:

```
'
public class Program
{
    public static void Main()
    {
        MyGame game = new MyGame();
        UDK.Game.Launch(game);
    }
}
'
```

Clase 21

Viernes, 20 Octubre 2023

Indice

Funciones complejas de contenedores

- Es correcto separar el funcionamiento del programa en clases que realizan labores mas concretas.
- La función Draw() le pide al personaje que se pinte, según su propia función Draw() definida en su clase.
- Hasta ahora tenemos las siguientes funciones:
  Program -> MyGame -> World -> Character // Utils
- Si tenemos una 'List<Character>' y la hacemos privada, hay dos funciones que son mas o menos obligatorias:
  'GetCharacterCount(), que devuelve el número de personajes'.
  'GetCharacterAt(index)', que devuelve el personaje en la posición indicada.
- Es MUY IMPORTANTE comprobar que la lista non se acceda con valores equivocados.
- A partir de ahora utilizaremos la clase 'RECTANGLE' para definir la forma y limites del personaje.
  Sus atributos son 'x,y,width,height'.
- Podemos definir si los cuerpos se interesectan o no a partir de las ecuaciones de intersección.

Clase 22

Lunes, 23 Octubre 2023

Indice

Clase practica

- Clase práctica para terminar el movimiento de los personajes

Clase 23

Martes, 24 Octubre 2023

Indice

Clase 'Jar'

```
'
       JAR
-----------------
-capacity: double
-quantity: double
-----------------
+SetQuantity(quantity): double
+GetQuantity(): double
+SetCapacity(value)
+GetCapacity(): double
+GetPercent(): double
+GetRemining(): double
+AddQuantity(value): void
'
```

- 'SetQuantity': Se le pasa una cantidad. Devuelve la cantidad que sobra
- 'GetPercent': Devuelve el porcentaje sobrante
- 'AddQuantity': Se le pasa la cantidad que añadimos a la jarra.

Clase 'Machine'

```
'
  COFFEEMACHINE
-----------------
-state: Estado
-----------------
+GetState(): Estado
+ChangeToNextState(): void
'
```

- dom(Estado): (esperando, procesando_Moneda, retirando_producto, devolviendo_cambio)
- En el constructor se inicia el objeto en estado 'esperando'.
- La forma óptima de diseñar la función Change() es con un 'Switch'

Clase 'Moneda'

```
'
    COINCHANGE
-----------------
+(static)ToNumber(coin:Moneda): int
+(static)ToMoneda(int:centims): Moneda
+(static)GetReturnChange(int:centims): List<Moneda>
'
```

- dom(Moneda): (Euro_50000,[...], Euro_1)
- Un 'enum' no puede empezar por número, ni símbolo.

Clase 24

Viernes, 27 Octubre 2023

Indice

Clase practica

- Un 'enum' es un 'int' camuflado. Podemos acceder a su indice si hacemos casting de (int)

```
'public static int MonedaToInt(Moneda moneda)
{
    return (int)Moneda;
}

public static Moneda IntToMoneda(int centims)
{
    return (Moneda)centims;
}
'
```

- Los atributos de clase se inicializan justo cuando se crea la clase que contiene el Objeto.
- Se puede modificar el **constructor** de la clase desde su propio **constructor estático**.

```
'public class CoinChange
{
    private static int _monedasChange;

    static CoinChange(int quantity)
    {
        _monedasChange = quantity;
    }
}
'
```

- El constructor de clase no lleva **public** y además es siempre **static**.

Clase 25

Lunes, 30 Octubre 2023

Indice

Clase practica 'DominoGame'

- Clase práctica para diseñar un juego de dominó.

Clase 26

Martes, 31 Octubre 2023

Indice

Clase 'CardGame'

- Clase práctica para diseñar un juego de cartas 'BlackJack'

Clase 27

Jueves, 2 Noviembre 2023

Indice

Clase ChessGame

- Clase práctica para terminar el movimiento de los personajes

Clase 28

Viernes, 3 Noviembre 2023

Indice

ChessGame

- Clase práctica para pintar el tablero
- Habitualmente, en las interfaces gráficas se coloca el origen (0,0) en la esquina superior izquierda. Las 'X' se incrementan hacia la derecha. Las'Y' se incrementan hacia abajo
- **FUNCIÓN:** IsFigureAbleToMove(int targetX, int targetY, ChessFigure figure)
- Se le pregunta a la función si es capaz de mover la pieza 'figure' hacia una nueva posición ' targetX, targetY'
- Recorro la lista de casillas y le pregunto si se puede mover un 'diferencial' respecto de su posición inicial

Clase 29

Lunes, 6 Noviembre 2023

Indice

Medieval Game

- Clase práctica para comenzar a diseñar la logica del juego medieval.
- **Jerarquía entre clases**
- Weapon->Composicion->Warrior->Composicion->Warzone->Composicion->Game
- **Clase Game:**
  Contiene las reglas del juego. El funcionamiento más general de todo el programa
- **Clase Warzone:**
  -ListWarriors: ListWarriors -width: int -height: int +CreateWarriors(count, type: int): ListWarrior +RemoveAt(index: int): void +MoveWarrior(Warrior, x, y : int): void
- **Clase Warrior**
  -x, -y : int -listWeapons: ListWeapon -life: int -accuraccy: double -winner: int -lucky: double -team: TeamType
  Dom (TeamType): (HUMAN, ELF, DWARF, ORC)
- **Clase Weapon:**
  -WeaponType: WeaponType +GetType(): WeaponType +GetDamage(): int +GetReloadTime(): int +CoolDown(): int
  Dom (WeaponType): (PUNCH, SWORD, MACE, BOW)

Clase 30

Martes, 7 Noviembre 2023

Indice

Medieval Game

- Dentro de la clase 'Weapon' es util la función 'GetDistance'. Es una función 'Static' que se resuelve con la función Math.Sqrt()
- Dentro de la clase 'Warzone', la función 'ExecuteTurn(Warzone)' recibe el parámetro de un 'Warzone warzone'. Esto permite que las clases que utilizan la función 'Execute', que son mas concretas, tengan la información de todo el programa, o de las clases más abstractas.
  Dentro de 'warzone' está toda la información que necesita, tales como listas, armas o movimientos.
- Dos funciones 'casi obligatorias' son el 'GetWarriorCount()' y 'GetWarriorAt(index)'. Estas funciones se diseñan casi al mismo tiempo.
- Son muy útiles las funciones que ordenan las listas de un modo determinado: por tipo, por distancia, por arma, por indice, etc...
- Otras funciones son: 'GetElementsAround()', 'GetWarriorAround()', 'GetWarriorsInside()', 'GetWarriorsSortByDistance()'
- Estas funciones se crean dentro de la clase 'Warzone' porque son de uso **'Global'** para todo el programa y para cada una de las clases mas concretas del programa.
- El diseño de lo **Abstracto hacia lo Concreto**, se denomina **Top-Down**.
- Al contrario, de lo **Concreto hacia lo Abstracto**, se denomina **Down-Top**.

Clase 31

Jueves, 9 Noviembre 2023

Indice

Clase de dudas generales

- Si la función no utiliza **objetos**, entonces es una funcion **STATIC**. Tambien es util entender que una función static no utiliza 'this'

Medieval Game

- Funciones útiles: Enemigos alrededor de una casilla, Amigos alrededor, Warrior alrededor, Ordenar las listas según cercania/lejania del Warrior
- Durante un turno se sucede una fase de **Movimiento** y una fase de **Ataque**
- Un método para implementar la IA de un programa es mediante el patrón de **'puntuación'**. La máquina aprende a utilizar uno u otro comportamiento, basandose en la puntuacion de los distintos metodos.
- Una forma de utilizar el framework de diseño visual es mediante el uso de un contador de frames. Se decide que realice una acción cuando llega a un número determinado de frames. Al final del bucle el contador se iguala a 0.

Clase 32

Lunes, 13 Noviembre 2023

Indice

Recordatorios

- Habitualmente una función de objeto, que lleva **'void'** no tiene parametros de entrada.
- Al revés, una función de clase, que lleva **'static'** recibe todos los datos de entrada por parámetros.
- Utilizamos tres barras '///' para documentar funciones. Le indicamos el uso de la función, los parámetros que recibe y el retorno.
- La instrucción **'break'** coloca la linea de compilación fuera del bucle más inmediato. Los bucles son 'while' y 'for'.
- Podemos averiguar el valor numérico de un 'char' mediante un casting de 'int'.

```
charExample = 'b';
int characterNumber = (int)charExample;
```

A la inversa también funciona, averiguar qué char es un numero en concreto.

Clase 'Datetime'

- Diseño de una nueva clase, llamada **'Datetime'**. Tiene los siguientes atributos:

```
-day: int
-month: int
-year: int
-seconds: int
-minutes: int
-hours: int

+Datetime()
+Datetime(day, month, year)
+Datetime(day, month, year, seconds, minutes, hours)

+Clone(): Datetime
+Equals(Datetime): bool
+IsValid(): bool
+IsLeap(): bool
+IsLeap'static'(year: int): bool
+ToString(): string
+GetDaysCount'static'(year, month): int
+IncrementDay(): int
+IncrementSecond(): int
+GetDayOfWeek(): DayOfTheWeek
```

- Datetime mide los momentos en los que ocurre algún hecho.
- Las fechas se almacenan con enteros. Todas se miden contando los segundos desde la fecha: 1 de Enero de 1970
- Los constructores que mantienen un mismo orden son más accesibles.
- La función 'static' del año bisiesto requiere de un parametro, ya que es imposible acceder desde un objeto año.

Clase 33

Martes, 14 Noviembre 2023

Indice

Clase 'Datetime'

- No se debe utilizar un enum con los tipos primitivos, excepto con los 'string'
- Entra en el examen, contar segundos minutos horas o cambiar de moneda. Se utiliza bucles 'while'.