

Programación Gráfica II

8. Manejador de Modelos y Picking Up.

Objetivo

- Mostrar el Picking Up utilizando DarkGDK.
- Diseñar un manejador de modelos, para administrar de mejor manera cada uno de éstos.
- Identificar los principales problemas a la hora de trabajar con DarkGDK y sus posibles soluciones.

Picking Up

- Picking Up es la técnica de seleccionar un objeto 3D en pantalla utilizando el mouse.
- Como resultado podremos mover un objeto utilizando el mouse, o realizar cualquier tipo de acciones con éste objeto seleccionado.



Picking Up

- Verifica si en las coordenadas especificadas de la pantalla se encuentra un objeto (entre iObjectStart y iObjectEnd). Si lo hay retorna el id del más cercano, de lo contrario retorna 0.

```
int dbPickObject ( int iX, int iY, int iObjectStart, int iObjectEnd )
```

- Calcula una coordenada 3D desde una coordenada 2D de la pantalla a una profundidad definida.

```
void dbPickScreen ( int iX, int iY, float fDistance )
```

Picking Up

- Retorna la distancia hacia el objeto seleccionado.

```
float dbGetPickDistance ( void )
```

- Retorna las coordenadas relativas a la posición en el mundo de la cámara.

```
float dbGetPickVectorX ( void )
```

```
float dbGetPickVectorY ( void )
```

```
float dbGetPickVectorZ ( void )
```

Proyecto Guía

Proyecto Guía

- Lo que haremos será cambiar entre personajes utilizando la técnica de picking up. Así cuando se haga click sobre un modelo cambiaremos de personaje, pudiendo controlarlo.



Proyecto Guía

- Antes de continuar nos crearemos una entidad que nos ayudará con el manejo de los modelos: “ModelManager”, quién será el encargado de administrar los diferentes tipos de aliados y enemigos, o personajes que queramos incorporar.



Proyecto Guía

- Las funciones de ModelManager serán:
 - Crear nuevos modelos.
 - Eliminar (una vez que han muerto).
 - Verificar si contiene un modelo.
 - Obtener un modelo según un identificador.
 - Actualizar el loop cada uno de los modelos.

Proyecto Guía

- Para esto utilizaremos una estructura llamada Diccionario, ya que nos permite almacenar objetos según un identificador único.

```
ref class ModelManager : public GameComponent
{
private:
    Dictionary<int , League^>^ leagues;
    int iModelCount; // Contador, se utilizará para los id de los modelos

public:
    ModelManager(void);

    // Actualiza todos los modelos
    virtual void Update() override;

    // Adhiere, remueve y obtiene aliados y verifica si el contenedor contiene aliados
    void AddLeague(float x, float y, float z);
    void DeleteLeague(int iModel);
    League^ getLeague(int iModel);
    bool ConstainLeague(int iModel);
};
```

Proyecto Guía

- Para adherir nuevos Aliados a nuestra lista utilizamos la clonación, así copiamos todas las características de un modelo ya creado anteriormente, y lo adherimos a nuestra lista de aliados.

```
void ModelManager::AddLeague(float x, float y, float z)
{
    int iModel = iModelCount++;
    dbCloneObject(iModel, GameConstant::iModelPlayer);
    dbPositionObject(iModel, x, y, z);
    dbShowObject(iModel);

    League^ league = gcnew League(iModel);
    leagues->Add(iModel, league);
}
```

Proyecto Guía

- Las funciones de eliminar, verificar si contiene y retornar el modelo son fáciles, básicamente llamar a las mismas funciones del diccionario.

```
League^ ModelManager::getLeague(int iModel)
{
    League^ league;
    leagues->TryGetValue( iModel, league);
    return league;
}
```

```
void ModelManager::DeleteLeague(int iModel)
{
    leagues->Remove(iModel);
}

bool ModelManager::ConstainLeague(int iModel)
{
    return leagues->ContainsKey(iModel);
}
```

Proyecto Guía

- Finalmente necesitamos que se actualicen todos los modelos que están vivos (o muertos-vivos si son zombies)

```
void ModelManager::Update()
{
    for each (League^ league in leagues->Values)
    {
        if(alive)
            league->Update();
    }
}
```

Proyecto Guía

- Ahora a lo que vinimos:
 - Cargaremos unos 20 personajes.
 - Y al hacer click con el mouse verificaremos si estamos sobre un personaje, de estarlo éste será nuestro player.



Proyecto Guía

```
void GameLevel::Initialize()
{
    dbLoadObject ("Chainsaw Brute.x" , GameConstant::iModelPlayer);
    dbLoadImage("Chainsaw Brute2_D.dds", GameConstant::iImagePlayer);
    dbTextureObject(GameConstant::iModelPlayer, GameConstant::iImagePlayer);
    dbHideObject(GameConstant::iModelPlayer);

    float distance = 150.0f;

    for(float i=0.0f; i<360.0f; i += 20.0f)
    {
        float x = distance * dbSin(i);
        float z = distance * dbCos(i);

        modelMgr->AddLeague(x, 0.0f, z);
    }

    player->IObject = GameConstant::iModelStartLeague;

    ...
}
```

Cargamos los personajes.

Proyecto Guía

```
void GameLevel::Update()
{
    if(dbMouseClicked())
    {
        int picking = dbPickObject (dbMouseX(), dbMouseY(), 100, 132);

        if(picking != 0 && modelMgr->ConstainLeague(picking))
        {
            player->IObject = picking;
        }
    }

    ...
}
```

Realizamos el Picking Up

Problemas con DarkGDK

Problemas con DarkGDK

- Mediante esta experiencia pudimos observar 3 problemas principales con la rapidez de la aplicación.
- 1. Al parecer DarkGDK utiliza como estructura de datos un diccionario, y como no es orientado a objetos no podemos acceder directamente a cada uno de los atributos de cada modelo; por lo que se debe acceder por medio de funciones, para lo cual el motor debe recorrer entre sus modelos hasta encontrar el que se desea y devolver el atributo señalado !!.

Problemas con DarkGDK

1. Posible Solución:

- Duplicar la información, esto quiere decir que cada clase que represente a un modelo, como un aliado o un enemigo debería guardar los atributos de éste en variables propias, como la posición actual, angulos de rotación actual, tamaño, etc.

Problemas con DarkGDK

2. Al aumentar el número de objetos, los FPS disminuyen.
 - Esto es obvio, y para todos los motores sucede lo mismo.

2. Solución:

- No sobrecargar de modelos, e irlos construyendo de forma dinámica, para esto generalmente se utilizan habitaciones, o simplemente la distancia al player; así cuando los enemigos ya no tienen visibilidad del player eliminarlos!!!.

Problemas con DarkGDK

3. Al aumentar el procesamiento de los objetos, los FPS disminuyen, esto lo acabamos de observar con las animaciones de los personajes.
2. Solución:
 - Realizar todo tipo de procesamiento sólo si el personaje está en un área visible de la pantalla. Además algunos procesamientos podemos no realizarlos en cada loop, ganando FPS 😊

Visibilidad en DarkGDK

- Para determinar si un objeto es visible en pantalla tenemos el siguiente comando:

```
int dbObjectInScreen ( int iObject )
```

- Y para obtener las coordenada en pantalla de dicho objeto:

```
int dbObjectScreenX ( int iObject )
```

```
int dbObjectScreenY ( int iObject )
```



Cuida los FPS, y utiliza diversas técnicas para que no disminuyan, ya que la jugabilidad va de la mano con ellos.