SDL:Tutorials:Simple Engine Framework

From GPWiki

Contents

Description

A solid game engine is key to succeeding in building a complete game. Having a flexible and solid framework powering your game makes your overall game easier and faster to code, read and understand, and also less error-prone and more stable.

I'm a very big fan of OOP (which is Object Oriented Programming for those of you living under a rock), so my approach to building a solid game framework is creating an engine class that handles the general aspects of the game. In this case, I've chosen the path of simplicity and gone with virtual functions provided and called by the engine. To use the engine just involves creating a new engine class and inheriting properties of the original engine class. As the virtual functions are overloaded by the new engine class, they will be called by the original engine framework, thus creating event-type functions for doing calculations, rendering etc. at the right times.

This engine is by *no means* perfect or complete, and it's still a work in progress. But it's simple, very flexible, works well and is easily extendable. So I thought I'd share it with you to learn from/laugh at.

Engine.h

File: **Engine.h**

```
#ifndef ENGINE_H
#define ENGINE_H

#include "SDL.h"

/** The base engine class. **/
class CEngine
{
private:
        /** Last iteration's tick value **/
        long m_lLastTick;

        /** Window width **/
        int m_iWidth;
        /** Window height **/
        int m_iHeight;

        /** Has quit been called? **/
        bool m_bQuit;

        /** The title of the window **/
        const char* m_czTitle;

        /** Screen surface **/
        SDL_Surface* m_pScreen;

        /** Is the window minimized? **/
        bool m_bMinimized;

        /** Variables to calculate the frame rate **/
        /** Tick counter. **/
        int m_iFPSTickCounter;

        /** Frame rate counter. **/
        int m_iFPSCounter;

        /** Stores the last calculated frame rate. **/
        int m_iCurrentFPS;

protected:
        void DoThink();
        void DoRender();

        void SetSize(const int& iWidth, const int& iHeight);

        void HandleInput();

public:
        CEngine();
        virtual ~CEngine();

        void Init();
        void Start();

        /** OVERLOADED - Data that should be initialized when the application starts. **/
        virtual void AdditionalInit     () {}

        /** OVERLOADED - All the games calculation and updating.
                @param iElapsedTime The time in milliseconds elapsed since the function was called last.
        **/
        virtual void Think              ( const int& iElapsedTime ) {}
        /** OVERLOADED - All the games rendering.
                @param pDestSurface The main screen surface.
        **/
        virtual void Render             ( SDL_Surface* pDestSurface ) {}

        /** OVERLOADED - Allocated data that should be cleaned up. **/
        virtual void End                () {}

        /** OVERLOADED - Window is active again. **/
        virtual void WindowActive       () {}

        /** OVERLOADED - Window is inactive. **/
        virtual void WindowInactive     () {}


        /** OVERLOADED - Keyboard key has been released.
                @param iKeyEnum The key number.
        **/
        virtual void KeyUp              (const int& iKeyEnum) {}

        /** OVERLOADED - Keyboard key has been pressed.
                @param iKeyEnum The key number.
        **/
        virtual void KeyDown            (const int& iKeyEnum) {}


        /** OVERLOADED - The mouse has been moved.
                @param iButton  Specifies if a mouse button is pressed.
                @param iX       The mouse position on the X-axis in pixels.
                @param iY       The mouse position on the Y-axis in pixels.
                @param iRelX    The mouse position on the X-axis relative to the last position, in pixels.
                @param iRelY    The mouse position on the Y-axis relative to the last position, in pixels.

                @bug The iButton variable is always NULL.
        **/
        virtual void MouseMoved         (const int& iButton,
                                         const int& iX,
                                         const int& iY,
                                         const int& iRelX,
                                         const int& iRelY) {}

        /** OVERLOADED - A mouse button has been released.
                @param iButton  Specifies if a mouse button is pressed.
                @param iX       The mouse position on the X-axis in pixels.
                @param iY       The mouse position on the Y-axis in pixels.
                @param iRelX    The mouse position on the X-axis relative to the last position, in pixels.
                @param iRelY    The mouse position on the Y-axis relative to the last position, in pixels.
        **/

        virtual void MouseButtonUp      (const int& iButton,
                                         const int& iX,
                                         const int& iY,
                                         const int& iRelX,
                                         const int& iRelY) {}

        /** OVERLOADED - A mouse button has been pressed.
                @param iButton  Specifies if a mouse button is pressed.
                @param iX       The mouse position on the X-axis in pixels.
                @param iY       The mouse position on the Y-axis in pixels.
                @param iRelX    The mouse position on the X-axis relative to the last position, in pixels.
                @param iRelY    The mouse position on the Y-axis relative to the last position, in pixels.
        **/
        virtual void MouseButtonDown    (const int& iButton,
                                         const int& iX,
                                         const int& iY,
                                         const int& iRelX,
                                         const int& iRelY) {}

        void            SetTitle        (const char* czTitle);
        const char*     GetTitle        ();

        SDL_Surface*    GetSurface      ();

        int             GetFPS          ();
};

#endif // ENGINE_H
```

Engine.cpp

File: **Engine.cpp**

```
#include "Engine.h"
#include <windows.h> // For the WaitMessage() function.

/** Default constructor. **/
CEngine::CEngine()
{
	m_lLastTick		= 0;
	m_iWidth		= 800;
	m_iHeight		= 600;
	m_czTitle		= 0;

	m_pScreen		= 0;

	m_iFPSTickCounter	= 0;
	m_iFPSCounter		= 0;
	m_iCurrentFPS		= 0;

	m_bMinimized		= false;
}

/** Destructor. **/
CEngine::~CEngine()
{
	SDL_Quit();
}

/** Sets the height and width of the window.
	@param iWidth The width of the window
	@param iHeight The height of the window
**/
void CEngine::SetSize(const int& iWidth, const int& iHeight)
{
	m_iWidth  = iWidth;
	m_iHeight = iHeight;
	m_pScreen = SDL_SetVideoMode( iWidth, iHeight, 0, SDL_SWSURFACE );
}

/** Initialize SDL, the window and the additional data. **/
void CEngine::Init()
{
	// Register SDL_Quit to be called at exit; makes sure things are cleaned up when we quit.
	atexit( SDL_Quit );

	// Initialize SDL's subsystems - in this case, only video.
	if ( SDL_Init( SDL_INIT_VIDEO ) < 0 )
	{
		fprintf( stderr, "Unable to init SDL: %s\n", SDL_GetError() );
		exit( 1 );
	}

	// Attempt to create a window with the specified height and width.
	SetSize( m_iWidth, m_iHeight );

	// If we fail, return error.
	if ( m_pScreen == NULL )
	{
		fprintf( stderr, "Unable to set up video: %s\n", SDL_GetError() );
		exit( 1 );
	}

	AdditionalInit();
}

/** The main loop. **/
void CEngine::Start()
{
	m_lLastTick = SDL_GetTicks();
	m_bQuit = false;
```

```
        // Main loop: loop forever.
        while ( !m_bQuit )
        {
                // Handle mouse and keyboard input
                HandleInput();

                if ( m_bMinimized ) {
                        // Release some system resources if the app. is minimized.
                        WaitMessage(); // pause the application until focus in regained
                } else {
                        // Do some thinking
                        DoThink();

                        // Render stuff
                        DoRender();
                }
        }

        End();
}

/** Handles all controller inputs.
        @remark This function is called once per frame.
**/
void CEngine::HandleInput()
{
        // Poll for events, and handle the ones we care about.
        SDL_Event event;
        while ( SDL_PollEvent( &event ) )
        {
                switch ( event.type )
                {
                case SDL_KEYDOWN:
                        // If escape is pressed set the Quit-flag
                        if (event.key.keysym.sym == SDLK_ESCAPE)
                        {
                                m_bQuit = true;
                                break;
                        }

                        KeyDown( event.key.keysym.sym );
                        break;

                case SDL_KEYUP:
                        KeyUp( event.key.keysym.sym );
                        break;

                case SDL_QUIT:
                        m_bQuit = true;
                        break;

                case SDL_MOUSEMOTION:
                        MouseMoved(
                                event.button.button,
                                event.motion.x,
                                event.motion.y,
                                event.motion.xrel,
                                event.motion.yrel);
                        break;

                case SDL_MOUSEBUTTONUP:
                        MouseButtonUp(
                                event.button.button,
                                event.motion.x,
                                event.motion.y,
                                event.motion.xrel,
                                event.motion.yrel);
                        break;

                case SDL_MOUSEBUTTONDOWN:
                        MouseButtonDown(
                                event.button.button,
                                event.motion.x,
                                event.motion.y,
                                event.motion.xrel,
                                event.motion.yrel);
                        break;

                case SDL_ACTIVEEVENT:
                        if ( event.active.state & SDL_APPACTIVE ) {
                                if ( event.active.gain ) {
                                        m_bMinimized = false;
                                        WindowActive();
                                } else {
                                        m_bMinimized = true;
                                        WindowInactive();
                                }
                        }
                        break;
                } // switch
        } // while (handling input)
}

/** Handles the updating routine. **/
void CEngine::DoThink()
{
        long iElapsedTicks = SDL_GetTicks() - m_lLastTick;
        m_lLastTick = SDL_GetTicks();

        Think( iElapsedTicks );

        m_iFPSTickCounter += iElapsedTicks;
}

/** Handles the rendering and FPS calculations. **/
void CEngine::DoRender()
{
        ++m_iFPSCounter;
        if ( m_iFPSTickCounter >= 1000 )
        {
                m_iCurrentFPS = m_iFPSCounter;
                m_iFPSCounter = 0;
                m_iFPSTickCounter = 0;
        }

        SDL_FillRect( m_pScreen, 0, SDL_MapRGB( m_pScreen->format, 0, 0, 0 ) );

        // Lock surface if needed
        if ( SDL_MUSTLOCK( m_pScreen ) )
                if ( SDL_LockSurface( m_pScreen ) < 0 )
                        return;

        Render( GetSurface() );

        // Unlock if needed
        if ( SDL_MUSTLOCK( m_pScreen ) )
                SDL_UnlockSurface( m_pScreen );

        // Tell SDL to update the whole gScreen
        SDL_Flip( m_pScreen );
}

/** Sets the title of the window
        @param czTitle A character array that contains the text that the window title should be set to.
**/
void CEngine::SetTitle(const char* czTitle)
{
        m_czTitle = czTitle;
        SDL_WM_SetCaption( czTitle, 0 );
}

/** Retrieve the title of the application window.
        @return The last set windows title as a character array.
        @remark Only the last set title is returned. If another application has changed the window title, then
that title won't be returned.
**/
const char* CEngine::GetTitle()
{
        return m_czTitle;
}
```

```
/** Retrieve the main screen surface.
        @return A pointer to the SDL_Surface surface
        @remark The surface is not validated internally.
**/
SDL_Surface* CEngine::GetSurface()
{
        return m_pScreen;
}

/** Get the current FPS.
        @return The number of drawn frames in the last second.
        @remark The FPS is only updated once each second.
**/
int CEngine::GetFPS()
{
        return m_iCurrentFPS;
}
```

Example Usage

Example usage of the framework using the virtual functions of the original class. To use the example, just replace the "to do" comments with the code you wish, and it will be automatically executed.

File: **Main.cpp**

```
#include "Engine.h"
#include <stdlib.h>

class CMyEngine: public CEngine
{
public:
        void AdditionalInit ();
        void Think          ( const int& iElapsedTime );
        void Render         ( SDL_Surface* pDestSurface );

        void KeyUp          (const int& iKeyEnum);
        void KeyDown        (const int& iKeyEnum);

        void MouseMoved     (const int& iButton,
                             const int& iX,
                             const int& iY,
                             const int& iRelX,
                             const int& iRelY);

        void MouseButtonUp  (const int& iButton,
                             const int& iX,
                             const int& iY,
                             const int& iRelX,
                             const int& iRelY);

        void MouseButtonDown(const int& iButton,
                             const int& iX,
                             const int& iY,
                             const int& iRelX,
                             const int& iRelY);

        void WindowInactive();
        void WindowActive();

        void End();
};


// Entry point
int main(int argc, char* argv[])  // <- this must match exactly, since SDL rewrites it
```

```
{
        CMyEngine Engine;

        Engine.SetTitle( "Loading..." );
        Engine.Init();

        Engine.SetTitle( "SDL Testing!" );
        Engine.Start();

        Engine.SetTitle( "Quitting..." );

        return 0;
}

void CMyEngine::AdditionalInit()
{
        // Load up additional data
}

void CMyEngine::Think( const int& iElapsedTime )
{
        // Do time-based calculations
}

void CMyEngine::Render( SDL_Surface* pDestSurface )
{
        // Display slick graphics on screen
}

void CMyEngine::KeyDown(const int& iKeyEnum)
{
    switch (iKeyEnum)
    {
    case SDLK_LEFT:
      // Left arrow pressed
      break;
    case SDLK_RIGHT:
      // Right arrow pressed
      break;
    case SDLK_UP:
      // Up arrow pressed
      break;
    case SDLK_DOWN:
      // Down arrow pressed
      break;
    }
}


void CMyEngine::KeyUp(const int& iKeyEnum)
{
        switch (iKeyEnum)
        {
        case SDLK_LEFT:
          // Left arrow released
          break;
        case SDLK_RIGHT:
          // Right arrow released
          break;
        case SDLK_UP:
          // Up arrow released
          break;
        case SDLK_DOWN:
          // Down arrow released
          break;
        }
}

void CMyEngine::MouseMoved(const int& iButton,
                           const int& iX,
                           const int& iY,
                           const int& iRelX,
                           const int& iRelY)
{
        // Handle mouse movement
        // iX and iY are absolute screen positions
        // iRelX and iRelY are screen position relative to last detected mouse movement
}

void CMyEngine::MouseButtonUp(const int& iButton,
                              const int& iX,
                              const int& iY,
                              const int& iRelX,
                              const int& iRelY)
{
        // Handle mouse button released
}

void CMyEngine::MouseButtonDown(const int& iButton,
                                const int& iX,
                                const int& iY,
                                const int& iRelX,
                                const int& iRelY)
{
        // Handle mouse button pressed
}

void CMyEngine::WindowInactive()
{
        // Pause game
}

void CMyEngine::WindowActive()
{
        // Un-pause game
}


void CMyEngine::End()
{
        // Clean up
}

// compile on mac os x:
// g++ -Wall -lSDLmain -lSDL Engine.cpp Main.cpp -framework Cocoa
```

By *Anders "Sion" Nissen*

Added 5. March 2005

Last updated 14. August 2005

Retrieved from "http://gpwiki.org/index.php/SDL:Tutorials:Simple_Engine_Framework"

Categories: C and SDL | All C articles | All SDL articles