I znowu możemy być razem :). Wiem, ze czekacie z niecierpliwością na kolejne odcinki kursu, wiec bez zbędnego jęczenia zaczynajmy bo świat 3D już czeka. Dzisiaj powiemy sobie co to takiego jest timer.

Timer

Jak sama nazwa wskazuje timer to cos co ma związek z czasem. Wielu z was już pewnie zachodzi w głowę jak to działa i po co w ogóle nam taki timer. W aplikacjach DirectX-a przeważnie nie będziemy wykorzystywali timera, ponieważ to poważnie spowalnialniałoby nam obliczenia. Rendering scen 3D odbywa się w każdej możliwie wolnej chwili, kiedy nasza aplikacja nie jest zajęta przetwarzaniem innych komunikatów (od myszki, klawiatury itp.), tak wiec odbywa się to niemalże bezpośrednio w pętli obsługi komunikatów (jak przyjdzie co do czego to pokaże jak to zrobić). Po co wiec nam taki timer ? Już podaje najprostszy z możliwych przykładów ;). Kiedyś byliśmy z kolega na tyle złośliwi, że podesłaliśmy koleżance program, który oprócz wyświetlania napisu "I love you" ;)), robił też inną, bardzo brzydką rzecz. Owa koleżanka, jako, że była bardzo spragniona stosów maili bez namysłu otworzyła nasz program no i mało nie spadla z krzesła ze szczęścia. Początkowo nie zauważyła, ze w wyniku perfidnej działalności naszego programu, jakiekolwiek okienko by nie otworzyła (mowa o edytorze) wszędzie pojawiała się literka "c" co sekundę nowa. Wyobraźcie sobie jej frustracje przy próbie wprowadzenia jakiegoś polecenia w Unix'ie :), he, he, to dopiero może nauczyć szybkiego pisania na klawiaturze :)... Po kilku godzinach znęcania zlitowaliśmy się nad nią :). Ale jak to się stało, ze ta literka się pojawiała co sekundę???

Otóż Windows, wśród wielu pożytecznych rzeczy daje nam do ręki coś takiego jak timer. Jest to nic innego jak cos w rodzaju stopera liczącego określony czas. Na początku programu ustawiamy sobie taki stoper na odpowiednią wartość, ustalamy mu numer, piszemy obsługę odpowiedniego komunikatu i do dzieła. Napisałem o numerze timera, dlaczego ? Ponieważ takich timerów możemy sobie ustawić w Windowsie bardzo, bardzo dużo, ale nie radziłbym od razu zapychać systemu, bo może się to źle skończyć (jak powszechnie wiadomo). My przeważnie potrzebujemy w naszym programie jeden, dwa to już aż nadto.

Jak działa taki timer ? Otóż po odliczeniu określonego czasu wykonywana jest obsługa komunikatu **WM_TIMER** (oczywiście w pętli obsługi komunikatów). Zostaną wykonane wszystkie funkcje zawarte w obsłudze tego komunikatu, potem nastąpi powtórne załadowanie licznika i zostanie odmierzona kolejna przerwa. Tak właśnie "załatwiliśmy" brzydko mówiąc nasz koleżankę. Co sekundę wysyłaliśmy do każdego aktywnego okienka na jej pulpicie klawisz ze znakiem, tu dla przykładu "c". A jak to zrobić ? Nie ma chyba nic prostszego:

Kod

Otóż, jak powiedziałem na wstępie najpierw ustawiamy nasz timer. Najlepiej zrobić to gdzieś na starcie programu, ja stosuje do tego komunikat **WM_CREATE**. Za pomocą funkcji API wygląda to tak:

```
case WM_CREATE:
    SetTimer( hWnd, 1, 1, NULL );
    break;
```

i już. Tyle wystarczy aby nasz timer zaczął działać. Może pokrótce o parametrach funkcji **SetTimer()**. Pierwszy to oczywiście okno, dla którego ustawiamy nasz timer. Drugi to numer naszego timera. Nie zdażyło się nigdy abym musiał używać więcej niż trzy (ulepszona wersja programu dla koleżanki odpalała jej dodatkowo co minutę osławionego baranka ! :))). Trzeci parametr to czas jaki ma odmierzać nasz timer, który podaje się w milisekundach. W tym przypadku podanie wartości 1 oczywiście nie spowoduje, ze zegar będzie nam tykał co jedna ms. Jest to niemożliwe, gdyż działanie timera opiera się głownie o czasomierz sprzętowy (18,2 raza na sekundę) i ma rożne działanie w rożnych wersjach Windows ! W Win95 i Win98 jeśli dobrze pójdzie to komunikat timera będzie wywoływany co ok. 55 ms., a w NT jest trochę lepiej bo co ok. 10 ms. Dlaczego tak jest to już słodka tajemnica panów inżynierów z Microsoftu. Oczywiście jest to przypadek, ze tak powiem najlepszy, ponieważ jeśli nasz system będzie bardzo obciążony to komunikaty timera będą wywoływane w wolnych chwilach, wiec czas ten może się wydłużyć, a nawet zmieniać ! Ostatnim parametrem funkcji **SetTimer()** jest wskaźnik na tzw. funkcje timera. A co to takiego ? Już wyjaśniam. Jeśli ostatnim parametrem funkcji **SetTimer()** jest **NULL** znaczy to tyle, ze obsługa komunikatu **WM_TIMER** jest dokonywana w głównej procedurze obsługi komunikatów. Możemy zrobić to też w trochę inny sposób. Możemy podać funkcje (jej adres), która będzie wywoływana po każdym wyzerowaniu się licznika timera. I ten adres będzie właśnie czwartym parametrem funkcji **SetTimer()**. Funkcja obsługi timera powinna mieć atrybut **CALLBACK**, czyli taki sam jak główna procedura okna. Dlaczego ? Z bardzo prostego powodu - nie wiemy dokładnie kiedy nasz licznik timera się wyzeruje a co za tym idzie nigdzie w programie nie wywołujemy jawnie tej funkcji... jej wywołaniem zajmie się Windows, kiedy nadejdzie odpowiedni czas :).

Do zakończenia naszej perfidnej działalności powinniśmy jeszcze przy zamykaniu programu mówiąc niełądnie zabić nasz timer. Robi to funkcja:

```
KillTimer( HWDN, UINT )
```

Pobiera ona okno, w którym uruchomiono timer, oraz jego numer. Jej wywołanie powoduje zatrzymanie naszego zegarka i (co ważniejsze), nie zaśmiecanie naszego systemu. Timer powinniśmy zabijać w obsłudze komunikatu np. **WM_DESTROY**, no chyba, ze zamarzymy zatrzymać go sobie w dowolnym miejscu programu co oczywiście nie stanowi żadnego problemu.

WM_TIMER

Jeśli korzystamy z pierwszego sposobu ustawienia timera (czyli bez specjalnej funkcji obsługi timera) to obsługa komunikatu od naszego czasomierza zajmuje się procedura okna. Jeśli wiec pragniemy aby cos wykonywało się po odmierzeniu ustawionego przez nas czasu musimy w naszej procedurze okna napisać obsługę komunikatu **WM_TIMER**. Nie rożni się ona niczym szczególnym od innych komunikatów. Może tu być prawie dowolny kawałek kodu, który powiedzmy dla przykładu będzie coś rysował co określony czas albo w naszym złośliwym przykładzie symulował naciskanie klawiszy, poruszanie myszka albo robił jeszcze inne perfidne rzeczy ;). Dla chętnych i ambitnych eksperymentatorów przykład:

```
case WM_TIMER:
    swtich( wParam )
    {
        case 1:
            InvalidateRect();
            break;

        default:
            break;
    }
    break;
```

Co spowoduje taki kawałek kodu ? Niektórzy już pewnie wiedza, ale tak dla pewności... będzie on co ustawiony wcześniej czas wywoływał obsługę komunikatu **WM_PAINT**, czyli po prostu odmalowywał obszar unieważniony okna. Ale uwaga ! będzie to robił dla czasu ustawionego dla timera z numerem jeden. Instrukcja **switch** w obsłudze tego komunikatu decyduje o tym, który timer będzie odmierzał czas dla określonych funkcji. Numer timera jest przekazywany do funkcji w postaci parametru komunikatu wParam. Możemy ustawić więcej timerow instrukcja **SetTimer()** i obsługiwać je w procedurze okna, lub każdemu napisać własną procedurę obsługi. Procedura obsługi timera (czyli wywoływana przez Windows) a nie w obsłudze **WM_TIMER** powinna mieć taka ogólną postać:

```
void CALLBACK TimerProc( HWND hWnd, UINT message, UINT timerID, DWORD time )
```

hWnd to oczywiście nic innego jak uchwyt okna, który zostaje przekazany funkcji SetTimer() przy tworzeniu timera. Zmienna o nazwie "message" ma zawsze wartość **WM_TIMER**, ponieważ Windows przekazuje do tej funkcji jedynie komunikaty timera. Zmienna **timerID** to jak nietrudno się domyśleć numer naszego timera, który będzie wywoływał ta funkcje, natomiast **time** to wartość jaka upłynęła od czasu uruchomienia systemu Windows na danym komputerze podawana w milisekundach. Maksymalna wartością jaka może osiągnąć ta zmienna to 49.7 dnia :), po upływie tego czasu licznik jest zerowany i liczenie zaczyna się od początku... ale sami powiedzcie czy któryś Windows wytrzymał tyle bez zawieszenia się ??? :). Dla ciekawości mogę powiedzieć, że wartość te można pobrać za pomocą funkcji:

```
GetTickCount()
```

która zwróci ilość milisekund.

Dokladniej

Tak w skrócie wyglądają podstawy użycia timerów w Windows. Mowa tu oczywiście o timerach korzystających z przerwania sprzętowego. Jest także inna możliwość. Jeśli zależy nam na bardzo dokładnym liczeniu czasu to system daje nam do ręki funkcje:

```
QueryPerformanceCounter()
```

Funkcja ta zwraca ilość tyknieć zegara o określonej częstotliwości. Ma jednak dosyć istotna wadę - działa tylko na systemach wyposażonych w licznik umożliwiający bardzo dokładny pomiar czasu. Częstotliwość tego zegara można pobrać za pomocą funkcji:

```
QueryPerformanceFrequency()
```

Jeśli system wyposażony jest w taki licznik to wartość ta będzie niezerowa. Wykorzystanie tych funkcji daje bardzo dokładny pomiar czasu (z dużą rozdzielczością) ale jak nadmieniłem jest ograniczony możliwościami sprzętu, choć dzisiejszy sprzęt to pewnie nie takie rzeczy robić umie :).