Chapitre VIII : Timer

par Loka

Date de publication : 22/06/2006

Dernière mise à jour : 22/06/2006

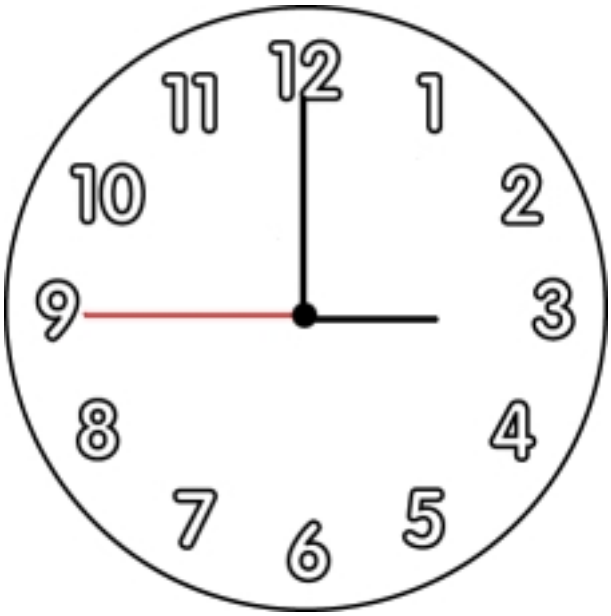
VIII - Timer
- VIII-A - Introduction
- VIII-B - La classe Timer
- VIII-C - Les méthodes de la classe Timer
- VIII-D - Première application : un chronomètre

VIII - Timer

VIII-A - Introduction

Disons que vous avez besoin d'attendre 30 secondes exactement mais que vous n'avez pas de chronomètre.

Si il y a une pendule sur le mur avec une aiguille pour les secondes, vous regardez le point de départ de l'aiguille des secondes :



Ensuite, il vous suffit d'attendre jusqu'à ce que 30 secondes passent depuis le point de départ :

La classe Timer que je vais vous présenter marche sur le même principe.

SDL a un timer de lancé et vous pouvez connaître le temps en millisecondes en utilisant la fonction **SDL_GetTicks()**.

Si vous devez chronométrer quelque chose pendant 1000 millisecondes, vous gardez en mémoire le temps de départ (le temps à partir duquel vous avez commencé à attendre) et vous attendez jusqu'à ce que la différence entre le temps de départ (gardé en mémoire) et le temps courant soit de 1000 millisecondes.

VIII-B - La classe Timer

Voici tout d'abord la classe Timer que nous allons utiliser :

classe Timer
```
//Le Timer
class Timer
{
    private:
    //Le temps quand le timer est lancé
    int startTicks;

    //Le temps enregistré quand le Timer a été mit en pause
    int pausedTicks;

    //Le status du Timer
    bool paused;
    bool started;

    public:
    //Initialise les variables
    Timer();

    //Les différentes actions du timer
    void start();
    void stop();
    void pause();
    void unpause();

    //recupére le nombre de millisecondes depuis que le timer a été lancé
    //ou récupére le nombre de millisecondes depuis que le timer a été mis en pause
    int get_ticks();

    //Fonctions de vérification du status du timer
    bool is_started();
    bool is_paused();
};
```

Cette classe possède les variables et les méthodes nécessaires au bon fonctionnement d'un Timer SDL.

Nous allons voir en détail chacune des méthodes qui la compose dans la prochaine section (VIII-C). Avant cela nous allons tout de même voir le constructeur de cette classe :

constructeur
```
Timer::Timer()
{
    //Initialisation des variables
    startTicks = 0;
    pausedTicks = 0;
    paused = false;
    started = false;
}
```

Comme vous pouvez le voir, le constructeur sert uniquement à initialiser les différentes variables, rien de plus.

VIII-C - Les méthodes de la classe Timer

Nous allons commencer par la méthode *void start()* :

start
```
void Timer::start()
{
    //On demarre le timer
    started = true;

    //On enlève la pause du timer
    paused = false;

    //On récupére le temps courant
    startTicks = SDL_GetTicks();
}
```

Nous démarrons donc le timer, nous mettons le statut *started* à *true*, nous enlevons la pause (même si celle-ci n'était pas activé) et ensuite nous récupérons le temps courant qui sera notre point de départ, que nous sauvegardons dans la variable *startTicks*.

Voici donc pour la méthode start de la classe Timer, nous allons donc continuer avec la méthode *void stop()* :

stop
```
void Timer::stop()
{
    //On stoppe le timer
    started = false;

    //On enlève la pause
    paused = false;
}
```

Lorsque nous stoppons notre timer, il suffit de mettre le statut *started* à *false* ainsi que d'enlever la pause.

Toujours simple, nous allons donc voir la méthode *int get_ticks()* :

get_ticks
```
int Timer::get_ticks()
{
    //Si le timer est en marche
    if( started == true )
    {
        //Si le timer est en pause
        if( paused == true )
        {
            //On retourne le nombre de ticks quand le timer a été mis en pause
            return pausedTicks;
        }
        else
        {
            //On retourne le temps courant moins le temps quand il a démarré
            return SDL_GetTicks() - startTicks;
        }
    }

    //Si le timer n'est pas en marche
    return 0;
}
```

Nous voici avec la fonction qui nous permet de récupérer le temps du timer.

Premièrement nous vérifions si le timer est en marche. Si c'est le cas, nous vérifions alors si celui-ci est en pause.

Si le timer est en pause (paused == true), nous retournons le temps que le timer avait lorsqu'il a été mit en pause. Nous verrons plus tard à propos de mettre en pause ou d'enlever la pause.

Si le timer n'est pas en pause, nous retournons la différence dans le temps depuis que le timer est lancé et le temps courrant.

Donc, si vous avez lancé le timer quand SDL_GetTicks() était à 10 000 et que maintenant SDL_GetTicks() est à 20 000, il retournera 10 000, c'est à dire que 10 secondes sont passés depuis que le timer à été lancé.

Si le timer n'a jamais été lancé, la fonction retourne 0.

Maintenant que nous avons vu une des fonctions les plus importante de notre classe, nous allons voir comment mettre en pause avec la méthode *void pause()* :

pause
```
void Timer::pause()
{
    //Si le timer est en marche et pas encore en pause
    if( ( started == true ) && ( paused == false ) )
    {
        //On met en pause le timer
        paused = true;

        //On calcul le pausedTicks
        pausedTicks = SDL_GetTicks() - startTicks;
    }
}
```

Maintenant si nous souhaitons mettre en pause le timer, nous vérifions d'abord si celui-ci est lancé ainsi que celui-ci ne soit pas déjà en pause.

Si nous pouvons mettre en pause le timer, nous mettons le statut *paused* à *true*, et nous gardons en mémoire le temps du timer dans *pausedTicks* (le temps exact où le timer à été mit en pause).

Donc si vous démarrez quand SDL_GetTicks() était à 5000, et que vous mettez en pause lorsqu'il est à 7000, *pausedTicks* sera égal à 2000.

Maintenant que nous savons mettre notre timer en pause, nous allons voir comment enlever la pause grâce à la méthode *void unpause()* :

unpause
```
void Timer::unpause()
{
    //Si le timer est en pause
    if( paused == true )
    {
        //on enlève la pause du timer
```

unpause
```
            paused = false;

            //On remet à zero le startTicks
            startTicks = SDL_GetTicks() - pausedTicks;

            //Remise à zero du pausedTicks
            pausedTicks = 0;
        }
    }
```

Lorsque nous voulons enlever la pause, nous vérifions d'abord si le timer est bien en pause.

Si c'est le cas, nous mettons le statut *paused* à *false*, puis nous mettons le temps de départ (symbolisé par la variable de classe *startTicks*) égal au temps courant moins le temps du timer quand celui-ci a été mit en pause.

Finalement , nous mettons la variable de classe *pausedTicks* à 0 car je préfère faire ainsi (il n'y a pas de vrais raisons).

Il nous reste à voir les méthodes *bool is_started()* et *bool is_paused()* qui nous permettent de vérifier, respectivement, si le timer est démarré et en pause :

is_started
```
    bool Timer::is_started()
    {
        return started;
    }
```

is_paused
```
    bool Timer::is_paused()
    {
        return paused;
    }
```

Je pense que vous n'avez pas vraiment besoin de moi pour voir comment ces deux méthodes fonctionnent.

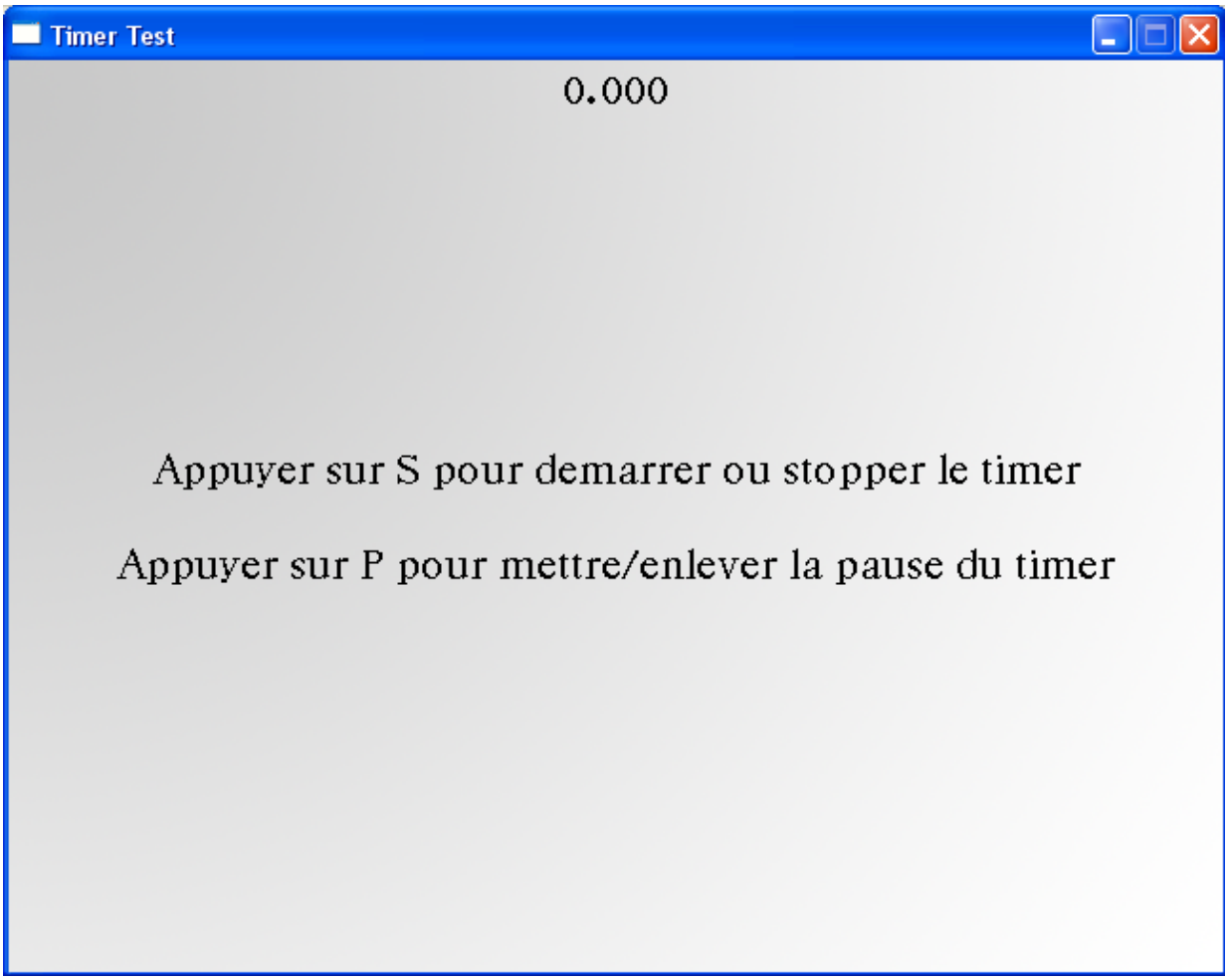
VIII-D - Première application : un chronomètre

Afin d'illustrer l'utilisation de notre classe Timer, nous allons "construire" un chronomètre.

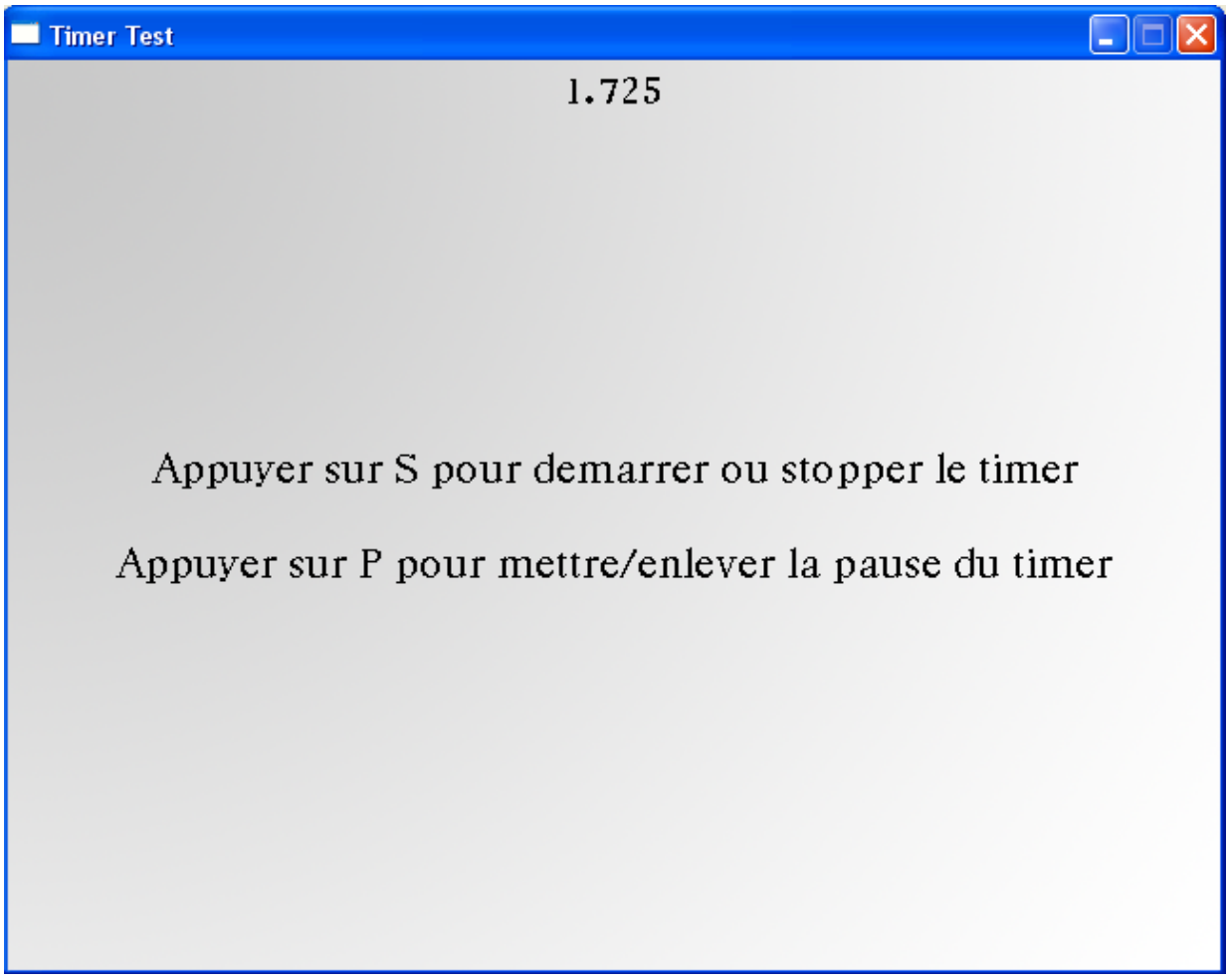
Ce chronomètre devra comporter les même fonctions qu'un chronomètre de base, c'est à dire pouvoir le demarrer (on lance le chronomètre), le stopper (on remet le temps à 0), le mettre en pause (garde le temps au moment de la mise en pause) et le relancer/enlever la pause (le chronomètre repart à partir du temps de pause).

Par défaut le chronomètre sera lancé au lancement de l'application.

Voici à quoi celui-ci ressemblera lorsque nous stopperons le chronomètre :



Et voici à quoi il ressemblera si nous le lançons, puis le mettons en pause au bout de 1,725 secondes :



Commençons donc notre programme, dans notre fonction main(), après initialisation de SDL et le chargement des différents fichiers dont on aura besoin, nous déclarons un objet de type Timer et nous mettons en place nos surfaces de message :

main
```
   //On construit le timer
       Timer myTimer;

       //On génére la surface de message
       startStop = TTF_RenderText_Solid( font, "Appuyer sur S pour demarrer ou stopper le timer",
   textColor );
       pause = TTF_RenderText_Solid( font, "Appuyer sur P pour mettre/enlever la pause du timer",
   textColor );
```

Avant d'entrer dans la boucle principale, il nous faut lancer le timer :

main
```
   //Mise en route du timer
       myTimer.start();
```

Ensuite vient donc la boucle principale avec la gestion du clavier, tout d'abord nous allons voir quoi faire quand la touche "s" a été pressée (la touche "s" sert démarrer ou stopper le timer) :

main
```
   //Tant que l'utilisateur n'a pas quitter
       while( quit == false )
       {
           //Tant qu'il y a un événement
           while( SDL_PollEvent( &event ) )
           {
               //Si une touche a été préssée
               if( event.type == SDL_KEYDOWN )
               {
                   //Si "s" a été préssé
                   if( event.key.keysym.sym == SDLK_s )
                   {
                       //Si le timer est en marche
                       if( myTimer.is_started() == true )
                       {
                           //On stoppe le timer
                           myTimer.stop();
                       }
                       else
                       {
                           //Mise en marche du timer
                           myTimer.start();
                       }
                   }
```

Donc, lorsque "s" est pressée, si le timer est en marche, on le stop, autrement on le démarre.

Pour la gestion de la pause, la touche associée est la touche "p", voici le code associé à la touche "p" :

main
```
   //Si "p" a été préssé
                   if( event.key.keysym.sym == SDLK_p )
                   {
                       //Si le timer est en pause
                       if( myTimer.is_paused() == true )
                       {
                           //On enlève la pause
                           myTimer.unpause();
                       }
                       else
                       {
                           //On met le timer en pause
                           myTimer.pause();
                       }
                   }
```

Donc lorsque la touche "p" est pressée, si le timer est en pause, on enlève la pause, autrement on met le timer en pause.

Pour la gestion des événements, il nous reste juste la fermeture de l'application qui se passe comme d'habitude.

nous allons terminer ce tutorial par l'application des différentes surfaces et l'affichage du temps.

Commençons par l'application des surfaces de fond et des messages :

main
```
   //On pose le fond
           apply_surface( 0, 0, background, screen );
```

main
```
          //On pose les surfaces
          apply_surface( ( SCREEN_WIDTH - startStop->w ) / 2, 200, startStop, screen );
          apply_surface( ( SCREEN_WIDTH - pause->w ) / 2, 250, pause, screen );
```

Si vous avez suivit les tutoriaux précèdent, vous devriez comprendre cette étape sans problèmes.

Voyons maintenant l'affichage du temps :

main
```
   //Une chaîne temporaire
          char time[ 64 ];

          //On convertis time en une chaîne de caractère (on le tronque aussi à trois chiffres après
   la virgule)
          sprintf( time, "%.3f", (float)myTimer.get_ticks() / 1000 );
```

Nous créons donc une chaîne temporaire, et nous mettons le temps du timer dans cette chaîne.

Vu que nous voulons le temps en secondes, nous convertissons le temps du timer dans en nombre flottant (float), puis nous divisons par 1000 (1000 millisecondes par secondes).

Maintenant il nous suffit de créer la surface depuis la chaîne qui garde le temps du timer :

main
```
   //mise en place de la surface time en texte
          seconds = TTF_RenderText_Solid( font, time, textColor );

          //On pose la surface
          apply_surface( ( SCREEN_WIDTH - seconds->w ) / 2, 0, seconds, screen );

          //On libère la surface de temps
          SDL_FreeSurface( seconds );

          //Mise à jour de l'écran
          if( SDL_Flip( screen ) == -1 )
          {
              return 1;
          }
```

La nouvelle surface montrant le temps du timer est posée sur l'écran, puis elle est libérée puisque nous en avons fini avec.

Après ça l'écran est mis à jour, et nous continuons la boucle principale, le temps est reposé puis libéré et ainsi de suite.

J'espère que vous avez compris le concept du timer et que vous entrevoyez les possibilités qu'il peut vous offrir, nous en verrons quelques unes dans les prochains tutoriaux en application avec de nouveaux concepts.

Version pdf (585 ko - pages)

Télécharger les sources (566 ko)