Objectifs

- Installer et tester l'environnement de développement
- Créer et compiler le projet
- Comprendre la gestion des graphismes
- Créer une bibliothèque

Il est impératif de valider chaque étape avant de passer à la suivante.

En cas de doute ou de difficulté, consultez votre enseignant.

Introduction

L'objectif de ce module est de concevoir une application de A à Z. Cette démarche permet de renforcer les compétences initiales en programmation, mais aussi de comprendre la conception d'un projet constitué de plusieurs fichiers et bibliothèques.

Installation des logiciels

L'ensemble des logiciels utilisés sont gratuits et peuvent fonctionner à la fois sous Windows et sous Linux. Toutefois, leur installation et utilisation est plus aisée sous Linux, c'est la raison pour laquelle nous allons travailler avec cet environnement. Ce projet a été conçu avec les versions suivantes :

- Linux Ubuntu 12.04 LTS
- Qt-creator 2.4.1
- SFML 1.6 (Simple and Fast Multimedia Library)

Qt-creator est notre environnement de développement, c'est avec ce logiciel que nous allons écrire le programme. SFML est une bibliothèque qui permet d'accéder facilement aux couches matérielles (graphismes, son, clavier, manettes, ...).

Pour installer Qt-creator, lancer un terminal (Ctrl-Alt-t) et exécuter la commande suivante :

```
$ sudo apt-get install qtcreator
```

Pour installer SFML, exécuter la commande suivante dans le terminal :

```
$ sudo apt-get install libsfml-dev
```

Création du projet

Lancer Qt-creator et créer un nouveau projet (Application Qt4 en console). Décocher Shadow Build, nommez votre projet **soccer** et sauvegardez-le dans un endroit où vous pourrez le retrouver les séances suivantes. Le chemin ne doit pas comporter d'espaces où de caractère spéciaux. N'oubliez pas que vous êtes seul responsable de la sauvegarde de vos fichiers.

Pour que le projet puisse intégrer la bibliothèque SFML, il va falloir spécifier au compilateur que nous allons l'utiliser. Editer le fichier **soccer.pro** de votre projet et ajouter les lignes suivantes à la fin :

```
LIBS    += -lsfml-graphics
LIBS    += -lsfml-window
LIBS    += -lsfml-system
```

Cela va indiquer au compilateur qu'il va devoir utiliser les bibliothèques SFML. Vous pouvez maintenant saisir et tester le début de votre projet avec le code suivant :

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>

int main()
{
        // Crée la fenêtre graphique : 1024x768 pixels 32bits par couleur
        sf::RenderWindow App(sf::VideoMode(1024, 768, 32), "Soccer GEII");

        // Boucle tant que l'application est ouverte
        while (App.IsOpened())
        {

            // ____________________________
            // ::: Gestion des événements :::

            // Effectue le traitement de chaque événement
            sf::Event Event;
            while (App.GetEvent(Event))
            {
                // Si la fenêtre est fermée, on ferme l'application
                if (Event.Type == sf::Event::Closed) App.Close();
            }

            // ____________________________
            // ::: Gestion des graphismes :::

            // Efface le fenêtre et définie la couleur de fond
            App.Clear(sf::Color(0x66, 0xCC, 0x66));

            // Rafraichie la fenêtre
            App.Display();
        }

        // L'application se termine sans erreur
        return EXIT_SUCCESS;
}
```

Avant de poursuivre, compiler et tester votre application. Une fenêtre avec un fond de couleur verte doit apparaître. La fermeture de la fenêtre doit entraîner la fin de l'exécution du programme.

Création des sprites

Un sprite est un terme technique dans le monde du jeu vidéo pour définir un élément graphique (généralement une image) qui peut se déplacer dans la scène (typiquement les personnages et les objets). Afin de pouvoir les utiliser facilement, une bibliothèque <sprite.h> vous est fournie. Copier les fichiers (sprite.h et sprite.cpp) dans votre répertoire de travail et ajouter les deux fichiers à votre projet. Pensez à ajouter la bibliothèque dans les inclusions de votre programme principal. Copier également le dossier /images dans votre répertoire de travail.

Pour créer un sprite, utiliser la ligne suivante :

```
Sprite Player1("images/player1.png");
```

Cette ligne est similaire à une création de variables (ici de type Sprite). Lors de la création, on spécifie en même temps le chemin et le nom du fichier qui contient l'image. Cette image sera automatiquement associée au sprite. Tout comme les variables, la déclaration doit être faîte une et une seule fois. Dans notre cas, il faut la mettre au début du programme principal. Notez que, pour l'instant, le sprite est uniquement créé en mémoire et n'est pas encore affiché.

Affichage des sprites

Avant d'afficher le sprite, il est important de comprendre quelques éléments liés au graphismes du programme principal. La ligne ci-dessous permet de créer une variable de type fenêtre de rendu (RenderWindow) qui s'appelle App. Ne perdez jamais de vu que App est une variable associée à la fenêtre.

```
sf::RenderWindow App(sf::VideoMode(1024, 768, 32), "Soccer GEII");
```

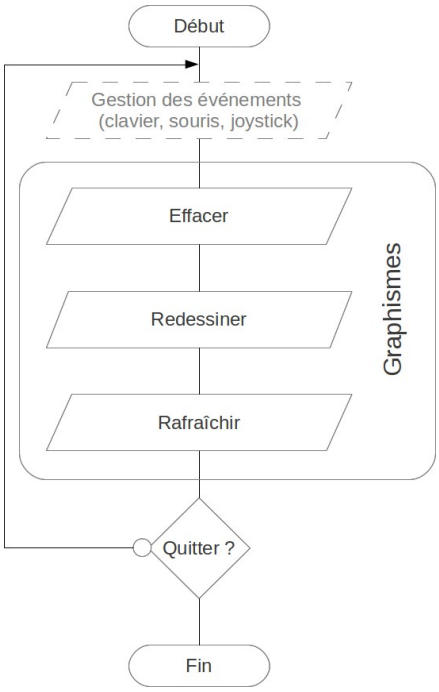
La ligne ci-dessous permet d'effacer le contenu de la fenêtre et de la remplir d'une couleur donnée (ici du vert) :

```
App.Clear(sf::Color(0x66, 0xCC, 0x66));
```

Pour éviter des saccades dans l'affichage, les commandes graphiques ne sont pas appliquées instantanément. Elles sont préparées en mémoire, puis toutes les modifications sont appliquées simultanément lors d'une action appelée « rafraîchissement ». Cela permet également de synchroniser l'affichage avec le balayage vertical de l'écran, ce qui évite le scintillement. Pour que les modifications soient visibles dans la fenêtre, il faut donc rafraîchir l'affichage avec la commande suivante :

```
App.Display();
```

Le programme est constitué d'une boucle principale et, à chaque cycle de cette boucle, l'écran est entièrement redessiné. Voici ci-dessous l'organigramme de notre application.



Pour afficher le sprite dans la fenêtre, utiliser la fonction Draw en spécifiant en paramètre le nom du sprite que vous souhaitez afficher. Insérer cette ligne entre l'effaçage et le rafraîchissement.

```
App.Draw(Player1);
```

Tester votre programme. Votre premier sprite doit s'afficher en haut à gauche de la fenêtre.

Déplacer un sprite

Il est possible de déplacer un sprite en spécifiant sa nouvelle position. Pour cela utiliser la fonction SetPosition :

```
Player1.SetPosition(200,100);
```

Modifier la position du sprite juste avant de l'afficher, testez.

Rotation d'un sprite

De la même façon, il est possible de déplacer un sprite en spécifiant son nouvel angle. Pour cela utiliser la fonction SetRotation :

```
Player1.SetRotation(45);
```

Effectuer une rotation (en plus du déplacement) du sprite avant de l'afficher, testez. Comme vous pouvez le constater, l'angle est définie en degrés. Cela n'est pas très pratique car les fonctions

mathématiques en C sont majoritairement en radians. Nous allons écrire une fonction de conversion des radians en degrés.

Bibliothèque d'outils

Afin de créer la fonction de conversion des radians en degrés, nous allons créer une bibliothèque. Créer deux fichiers dans votre répertoire de travailque vous nommerez tools.h et tools.cpp. Ajouter les fichiers au projet.

Recopier le contenu du fichier tools.h:

```
#ifndef TOOLS_H
#define TOOLS_H

// Fonction de conversion d'un angle (radians -> dégrées)
double Radians_To_Degrees(double Rad);

#endif // TOOLS_H
```

Les deux premières lignes et la dernière permettent d'éviter la redéfinition en cas d'inclusions multiples. Si plusieurs fichiers utilisent la librairie, les fonctions de cette dernière seront déclarées plusieurs fois ce qui ne sera pas permis par le compilateur. La première ligne teste l'existence d'une constante TOOLS_H. Si celle-ci n'existe pas, elle est créée (deuxième ligne) et les fonctions sont déclarées. Si cette constante existe déjà, cela signifie que les fonctions ont déjà été déclarée et que ce n'est plus à faire, on passe directement à la dernière ligne. Vous devrez utiliser ce méchanisme à chaque fois que vous créerez une nouvelle bibliothèque.

Recopier le contenu du fichier tools.cpp et compléter l'implémentation de la fonction. Il existe une constante M_PI qui représente le nombre pi, elle se trouve dans la bibliothèque math.h.

```
#include "tools.h"

// Fonction de conversion d'un angle (radians -> dégrées)
double Radians_To_Degrees(double Rad)
{
    // A compléter
}
```

Tester votre fonction avec la rotation suivante :

```
Player1.SetRotation(Radians_To_Degrees(3.1415));
```

Commentez, indentez et faites valider par l'enseignant.