Ogre Wiki

Support and community documentation for Ogre3D

This article will describe a simple technique which will help you to manage game states in OGRE, using the default rendering loop based on frame listeners. This technique is heavily inspired by the one described in the article Managing Game States in C++. As I won't give many explanations about the mechanics of this technique, I invite you to read this article if you want a more thoroughful analysis of the system.

Once you finish with the article about the basic design behind this Game Manager, there is a Game State Manager version which does not require the use of globals or singletons, but has its original design based here.

Note:
The code **does not work with the current Ogre version**. It was published for Ogre 1.0 and 1.2.
For usage with Ogre 1.4 look here.
For usage with the current Ogre version the code needs updates. If you update it, please share it with the community.

Table of contents

The GameState class

First, you'll need to create the file **GameState.h** and put it in your include or src directory, depending on how your source tree is structured. Here are the contents of this file :

```
#ifndef GameState_H
#define GameState_H

#include <OGRE/Ogre.h>

#include "GameManager.h"
class GameState
{
public:
    virtual void enter() = 0;
    virtual void exit() = 0;

    virtual void pause() = 0;
    virtual void resume() = 0;

    virtual void keyClicked(Ogre::KeyEvent* e) = 0;
    virtual void keyPressed(Ogre::KeyEvent* e) = 0;
    virtual void keyReleased(Ogre::KeyEvent* e) = 0;
    virtual bool frameStarted(const Ogre::FrameEvent& evt) = 0;
    virtual bool frameEnded(const Ogre::FrameEvent& evt) = 0;

    void changeState(GameState* state) { GameManager::getSingletonPtr()->chan
geState(state); }
    void pushState(GameState* state) { GameManager::getSingletonPtr()->pushSt
ate(state); }
    void popState() { GameManager::getSingletonPtr()->popState(); }
protected:
    GameState() { }
};

#endif
```

This is a basic game state class, which you'll need to inherit from for all the game states you want to handle in your game. I'll give an example of such game states later.

The InputManager class

The InputManager class, which is a singleton, will be the central point for input handling. Its role is to create an EventProcessor and expose an InputReader to the game states, which can be used for both buffered and unbuffered input. Unbuffered input will be used in the game itself and the buffered one will be use for states which should be treated as menus. More on that later when we come to the actual implementation of game states.

Definition of the class in **InputManager.h** :

```
#ifndef InputManager_H
#define InputManager_H

#include <OGRE/OgreSingleton.h>
#include <OGRE/OgreInput.h>

class InputManager : public Ogre::Singleton<InputManager>
{
public:
    InputManager(Ogre::RenderWindow* rwindow);
    virtual ~InputManager();
    inline Ogre::InputReader* getInputDevice() const { return mInputDevice; }
    inline Ogre::EventProcessor* getEventProcessor() const { return mEventPro
cessor; }
private:
    Ogre::EventProcessor* mEventProcessor;
    Ogre::InputReader* mInputDevice;
};

#endif
```

Implementation of the class in **InputManager.cpp** :

```
#include <OGRE/OgreEventProcessor.h>

#include "InputManager.h"

template<> InputManager* Ogre::Singleton<InputManager>::ms_Singleton = 0;

InputManager::InputManager(Ogre::RenderWindow* rwindow)
{
    mEventProcessor = new Ogre::EventProcessor();
    mEventProcessor->initialise(rwindow);
    mEventProcessor->startProcessingEvents();
    mInputDevice = mEventProcessor->getInputReader();
}

InputManager::~InputManager()
{
    if (mEventProcessor)
        delete mEventProcessor;
    assert(mInputDevice);
    Ogre::PlatformManager::getSingleton().destroyInputReader(mInputDevice);
}
```

The GameManager class

You'll need a game state manager to handle the game states and switch from one state to the other. This class, which is also a singleton, creates an InputManager and registers its EventProcessor to handle both buffered and unbuffered input. Here is the source code for the header file, named **GameManager.h** :

```
#ifndef GameManager_H
#define GameManager_H

#include <vector>
#include <OGRE/Ogre.h>
#include <OGRE/OgreEventListeners.h>
#include <OGRE/OgreSingleton.h>

#include "InputManager.h"

class GameState;

class GameManager : public Ogre::FrameListener, public Ogre::KeyListener,
    public Ogre::Singleton<GameManager>
{
public:
    GameManager();
    ~GameManager();
    void start(GameState* state);
    void changeState(GameState* state);
    void pushState(GameState* state);
    void popState();
    static GameManager& getSingleton(void);
    static GameManager* getSingletonPtr(void);
protected:
    Ogre::Root* mRoot;
    Ogre::RenderWindow* mRenderWindow;
    InputManager* mInputManager;

    void setupResources(void);
    bool configure(void);

    void keyClicked(Ogre::KeyEvent* e);
    void keyPressed(Ogre::KeyEvent* e);
    void keyReleased(Ogre::KeyEvent* e);
    bool frameStarted(const Ogre::FrameEvent& evt);
    bool frameEnded(const Ogre::FrameEvent& evt);
private:
    std::vector<GameState*> mStates;
};

#endif
```

And here is the source code for the main source file, named **GameManager.cpp** :

```
#include <OGRE/Ogre.h>

#include "GameManager.h"
#include "InputManager.h"
#include "GameState.h"

using namespace Ogre;

template<> GameManager* Singleton<GameManager>::ms_Singleton = 0;

GameManager::GameManager()
{
    mRoot = 0;
    mInputManager = 0;
}

GameManager::~GameManager()
{
    // clean up all the states
    while (!mStates.empty()) {
        mStates.back()->exit();
        mStates.pop_back();
    }

    if (mInputManager)
        delete mInputManager;

    if (mRoot)
        delete mRoot;
}

void GameManager::start(GameState* state)
{
    mRoot = new Root();

    if (!configure()) return;

        setupResources();

    mRoot->addFrameListener(this);

    mInputManager = new InputManager(mRoot->getAutoCreatedWindow());
    mInputManager->getEventProcessor()->addKeyListener(this);

    changeState(state);
```

```
    mRoot->startRendering();
}

void GameManager::changeState(GameState* state)
{
    // cleanup the current state
    if ( !mStates.empty() ) {
        mStates.back()->exit();
        mStates.pop_back();
    }

    // store and init the new state
    mStates.push_back(state);
    mStates.back()->enter();
}

void GameManager::pushState(GameState* state)
{
    // pause current state
    if ( !mStates.empty() ) {
        mStates.back()->pause();
    }

    // store and init the new state
    mStates.push_back(state);
    mStates.back()->enter();
}

void GameManager::popState()
{
    // cleanup the current state
    if ( !mStates.empty() ) {
        mStates.back()->exit();
        mStates.pop_back();
    }

    // resume previous state
    if ( !mStates.empty() ) {
        mStates.back()->resume();
    }
}

void GameManager::setupResources(void)
{
    // load resource paths from config file
    ConfigFile cf;
```

```
    cf.load("resources.cfg");

    // go through all settings in the file
    ConfigFile::SectionIterator seci = cf.getSectionIterator();

    String secName, typeName, archName;
    while (seci.hasMoreElements())
    {
        secName = seci.peekNextKey();
        ConfigFile::SettingsMultiMap *settings = seci.getNext();
        ConfigFile::SettingsMultiMap::iterator i;
        for (i = settings->begin() ; i != settings->end() ; ++i)
        {
            typeName = i->first;
            archName = i->second;
            ResourceGroupManager::getSingleton().addResourceLocation(
                archName, typeName, secName);
        }
    }
}

bool GameManager::configure(void)
{
    // load config settings from ogre.cfg
    if (!mRoot->restoreConfig())
    {
        // if there is no config file, show the configuration dialog
        if (!mRoot->showConfigDialog())
        {
            return false;
        }
    }

    // initialise and create a default rendering window
    mRenderWindow = mRoot->initialise(true);

    ResourceGroupManager::getSingleton().initialiseAllResourceGroups();

    return true;
}

void GameManager::keyClicked(KeyEvent* e)
{
    // call keyClicked of current state
    mStates.back()->keyClicked(e);
}
```

```
void GameManager::keyPressed(KeyEvent* e)
{
    // call keyPressed of current state
    mStates.back()->keyPressed(e);
}

void GameManager::keyReleased(KeyEvent* e)
{
    // call keyReleased of current state
    mStates.back()->keyReleased(e);
}

bool GameManager::frameStarted(const FrameEvent& evt)
{
    // call frameStarted of current state
    return mStates.back()->frameStarted(evt);
}

bool GameManager::frameEnded(const FrameEvent& evt)
{
    // call frameEnded of current state
    return mStates.back()->frameEnded(evt);
}

GameManager* GameManager::getSingletonPtr(void)
{
    return ms_Singleton;
}

GameManager& GameManager::getSingleton(void)
{
    assert(ms_Singleton);
    return *ms_Singleton;
}
```

Note that you normally won't need to modify the **GameState.h**, **GameManager.h**, **GameManager.cpp**, **InputManager.h** and **InputManager.cpp** files as all your code will usually reside in the classes derived from GameState.

Game states

I'll define three game states (**Intro**, **Play** and **Pause**) to show how this technique can be used, just like in the original source code for the article. Here is how these states are related to each other :

- in the **Intro** state, <SPACE> will lead to the **Play** state, <ESC> will terminate the application ;
- in the **Play** state, <ESC> will go back to the **Intro** state and <P> will go into the **Pause** state ;
- in the **Pause** state, another key press on <P> will return to the **Play** state.

There won't be a lot of things happening in each state, I'll only change the background color so you know in which state you are (red for **Intro**, blue for **Play** and green for **Pause**). You can easily extend the code from this base, adding support for scene nodes, entities, overlays, etc. The code should be quite self-explanatory, if in doubt, read the article for which I gave a link in the introduction to clear things up.

Here is the source code for the **Intro** state :

IntroState.h

```
#ifndef IntroState_H
#define IntroState_H

#include <OGRE/Ogre.h>

#include "GameState.h"

class IntroState : public GameState
{
public:
    void enter();
    void exit();

    void pause();
    void resume();

    void keyClicked(Ogre::KeyEvent* e);
    void keyPressed(Ogre::KeyEvent* e);
    void keyReleased(Ogre::KeyEvent* e);
    bool frameStarted(const Ogre::FrameEvent& evt);
    bool frameEnded(const Ogre::FrameEvent& evt);

    static IntroState* getInstance() { return &mIntroState; }
protected:
    IntroState() { }

    Ogre::Root *mRoot;
    Ogre::SceneManager* mSceneMgr;
    Ogre::Viewport* mViewport;
    Ogre::InputReader* mInputDevice;
    Ogre::Camera* mCamera;
    bool mExitGame;
private:
    static IntroState mIntroState;
};

#endif
```

IntroState.cpp

```
#include <OGRE/Ogre.h>
#include <OGRE/OgreKeyEvent.h>

#include "IntroState.h"
#include "PlayState.h"

using namespace Ogre;

IntroState IntroState::mIntroState;

void IntroState::enter()
{
    mInputDevice = InputManager::getSingletonPtr()->getInputDevice();
    mRoot = Root::getSingletonPtr();

    //should be for Ogre 1.2 createSceneManager(ST_GENERIC);
    mSceneMgr = mRoot->getSceneManager(ST_GENERIC);
    mCamera = mSceneMgr->createCamera("IntroCamera");
    mViewport = mRoot->getAutoCreatedWindow()->addViewport(mCamera);
    mViewport->setBackgroundColour(ColourValue(1.0, 0.0, 0.0));

    mExitGame = false;
}

void IntroState::exit()
{
    mSceneMgr->clearScene();
        //!! Note: This is supposed to be mSceneMgr->destroyAllCameras(); for
CVS head
    mSceneMgr->removeAllCameras();
    mRoot->getAutoCreatedWindow()->removeAllViewports();
}

void IntroState::pause()
{
}

void IntroState::resume()
{
}

void IntroState::keyClicked(KeyEvent* e)
{
}

void IntroState::keyPressed(KeyEvent* e)
```

```
{
    if (e->getKey() == KC_SPACE)
    {
        changeState(PlayState::getInstance());
    }

    if (e->getKey() == KC_ESCAPE)
    {
        mExitGame = true;
    }
}

void IntroState::keyReleased(KeyEvent* e)
{
}

bool IntroState::frameStarted(const FrameEvent& evt)
{
    return true;
}

bool IntroState::frameEnded(const FrameEvent& evt)
{
    if (mExitGame)
        return false;

    return true;
}
```

Here is the source code for the **Play** state :

PlayState.h

```
#ifndef PlayState_H
#define PlayState_H

#include <OGRE/Ogre.h>

#include "GameState.h"

class PlayState : public GameState
{
public:
    void enter();
    void exit();

    void pause();
    void resume();

    void keyClicked(Ogre::KeyEvent* e);
    void keyPressed(Ogre::KeyEvent* e);
    void keyReleased(Ogre::KeyEvent* e);
    bool frameStarted(const Ogre::FrameEvent& evt);
    bool frameEnded(const Ogre::FrameEvent& evt);

    static PlayState* getInstance() { return &mPlayState; }
protected:
    PlayState() { }

    Ogre::Root *mRoot;
    Ogre::SceneManager* mSceneMgr;
    Ogre::Viewport* mViewport;
    Ogre::InputReader* mInputDevice;
    Ogre::Camera* mCamera;
private:
    static PlayState mPlayState;
};

#endif
```

PlayState.cpp

```
#include <OGRE/Ogre.h>
#include <OGRE/OgreKeyEvent.h>

#include "PlayState.h"
#include "IntroState.h"
#include "PauseState.h"

using namespace Ogre;

PlayState PlayState::mPlayState;

void PlayState::enter()
{
    mInputDevice = InputManager::getSingletonPtr()->getInputDevice();
    mRoot = Root::getSingletonPtr();

    //should be for Ogre 1.2 createSceneManager(ST_GENERIC);
    mSceneMgr = mRoot->getSceneManager(ST_GENERIC);
    mCamera = mSceneMgr->createCamera("IntroCamera");
    mViewport = mRoot->getAutoCreatedWindow()->addViewport(mCamera);
    mViewport->setBackgroundColour(ColourValue(0.0, 0.0, 1.0));
}

void PlayState::exit()
{
    mSceneMgr->clearScene();
        //!! Note: This is supposed to be mSceneMgr->destroyAllCameras(); for
CVS head
    mSceneMgr->removeAllCameras();
    mRoot->getAutoCreatedWindow()->removeAllViewports();
}

void PlayState::pause()
{
}

void PlayState::resume()
{
    mViewport->setBackgroundColour(ColourValue(0.0, 0.0, 1.0));
}

void PlayState::keyClicked(KeyEvent* e)
{
}

void PlayState::keyPressed(KeyEvent* e)
```

```
{
    if (e->getKey() == KC_P)
    {
        pushState(PauseState::getInstance());
    }

    if (e->getKey() == KC_ESCAPE)
    {
        changeState(IntroState::getInstance());
    }
}

void PlayState::keyReleased(KeyEvent* e)
{
}

bool PlayState::frameStarted(const FrameEvent& evt)
{
    return true;
}

bool PlayState::frameEnded(const FrameEvent& evt)
{
    return true;
}
```

Here is the source code for the **Pause** state :

PauseState.h

```
#ifndef PauseState_H
#define PauseState_H

#include <OGRE/Ogre.h>

#include "GameState.h"

class PauseState : public GameState
{
public:
    void enter();
    void exit();

    void pause();
    void resume();

    void keyClicked(Ogre::KeyEvent* e);
    void keyPressed(Ogre::KeyEvent* e);
    void keyReleased(Ogre::KeyEvent* e);
    bool frameStarted(const Ogre::FrameEvent& evt);
    bool frameEnded(const Ogre::FrameEvent& evt);

    static PauseState* getInstance() { return &mPauseState; }
protected:
    PauseState() { }

    Ogre::Root *mRoot;
    Ogre::SceneManager* mSceneMgr;
    Ogre::Viewport* mViewport;
    Ogre::InputReader* mInputDevice;
    Ogre::Camera* mCamera;
private:
    static PauseState mPauseState;
};

#endif
```

PauseState.cpp

```
#include <OGRE/Ogre.h>
#include <OGRE/OgreKeyEvent.h>

#include "PauseState.h"
#include "PlayState.h"

using namespace Ogre;

PauseState PauseState::mPauseState;

void PauseState::enter()
{
    mInputDevice = InputManager::getSingletonPtr()->getInputDevice();
    mRoot = Root::getSingletonPtr();

    mViewport = mRoot->getAutoCreatedWindow()->getViewport(0);
    mViewport->setBackgroundColour(ColourValue(0.0, 1.0, 0.0));
}

void PauseState::exit()
{
}

void PauseState::pause()
{
}

void PauseState::resume()
{
}

void PauseState::keyClicked(KeyEvent* e)
{
}

void PauseState::keyPressed(KeyEvent* e)
{
    if (e->getKey() == KC_P)
    {
        popState();
    }
}

void PauseState::keyReleased(KeyEvent* e)
{
}
```

```
bool PauseState::frameStarted(const FrameEvent& evt)
{
    return true;
}

bool PauseState::frameEnded(const FrameEvent& evt)
{
    return true;
}
```

Note that all the input code related to buffered input should be in the **keyClicked**, **keyPressed** and **keyReleased** functions, using the **e->getKey()** call as shown in the sources. This is needed for everything related to menus or GUI.

Input management for the game itself (unbuffered input) should be handled in the **frameStarted** and **frameEnded** functions, using the **mInputDevice->isKeyDown()** call like this :

```
if (mInputDevice->isKeyDown(KC_RIGHT)) x++;
if (mInputDevice->isKeyDown(KC_LEFT)) x--;
if (mInputDevice->isKeyDown(KC_UP)) y++;
if (mInputDevice->isKeyDown(KC_DOWN)) y--;
```

This is what you will use to control a character on the screen or to shoot a bullet for example.

Gluing it all together

With all this files, we need a starting point for the application, it will be the **Main.cpp** file. It is very similar to the ones found in the OGRE samples. In this file, you'll only need to instantiate the game manager and start it in the first state.

Main.cpp

```
#include <OGRE/Ogre.h>

#include "GameManager.h"
#include "IntroState.h"

#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
#define WIN32_LEAN_AND_MEAN
#include "windows.h"

INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT)
#else
int main(int argc, char **argv)
#endif
{
    GameManager* game = new GameManager();

    try
    {
        // initialize the game and switch to the first state
        game->start(IntroState::getInstance());
    }
    catch (Ogre::Exception& e)
    {
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
        MessageBox(NULL, e.getFullDescription().c_str(), "An exception has oc
curred!", MB_OK | MB_ICONERROR | MB_TASKMODAL);
#else
        std::cerr << "An exception has occured: " << e.getFullDescription();
#endif
    }

    delete game;

    return 0;
}
```

(edited by stoneCold):

Adding Mouse Support

With some little changes to the InputManager class you can add mouse support to this tutorial which is needed in most applications.
It has been tested on Win32 only, but it should work well on other platforms too. (error reports are welcome)

Necessary Changes

- Add this line to **InputManager.h**

```
...
#include "OgrePlatformManager.h"
...
```

- and change the constructor of the InputManager (in **InputManager.cpp**) from this

```
InputManager::InputManager(Ogre::RenderWindow* window)
{
   mEventProcessor = new Ogre::EventProcessor();
   mEventProcessor->initialise(window);
   mEventProcessor->startProcessingEvents();

   //the old line
   mInputDevice = mEventProcessor->getInputReader();
}
```

to this

```
InputManager::InputManager(Ogre::RenderWindow* window)
{
   mEventProcessor = new Ogre::EventProcessor();
   mEventProcessor->initialise(window);
   mEventProcessor->startProcessingEvents();

   //the two new lines
   mInputDevice = Ogre::PlatformManager::getSingleton().createInputReader();
   mInputDevice->initialise(window, true, true);
}
```

- additionally you must "capture" the InputReader in the GameState where you want to read the mouse attributes

(be sure to do the "capture()" before accessing the mouse, else it will give you wrong results)

```
mInputDevice->capture();
```

Conclusion

This ends this short tutorial which described a way to handle game states within an OGRE

application. I hope it has been clear and that it will be useful for someone. For now it does only handle keyboard as an input device, I'll maybe add mouse and joystick support later if it fits well in this architecture. I am aware that this may not be the best way to handle game states within an application, and I'm open to any critics or suggestions about the way it was implemented.

- stateman-0.0.5.tar.gz - Source and autoconf/automake files for Linux (compatible with OGRE 1.0.4)
- statemanVC7.zip - Source and project file for Visual C++ 7 (old version, deprecated)
- statemanVC8.zip - Updated source and project file for Visual C++ 8 and Ogre 1.2.1 Dagon

---

Alias: Managing_Game_States_with_OGRE

Contributors to this page: eLeMenCy , Beauty , jacmoe and OgreWikiBot .
Page last modified on Sunday 14 of October, 2012 14:03:39 PDT by eLeMenCy.

Source History

Search by Tags

Search Wiki by Freetags

Latest Changes

1. Ogre 2.1 FAQ
2. ManualObject
3. Advanced Mogre Framework
4. OgreBites
5. Advanced Ogre Framework
6. Tutorials
7. Basic Tutorial 2
8. Basic Tutorial 1
9. Basic Tutorial Introduction
10. CMake Quick Start Guide

...more

Search

Find

Go

Advanced Search Help

Online Users

21 online users