|  |
| --- |
| TDE Game Engine |
| Code Samples |
| http://student.computing.dcu.ie/blogs/donneln7/ |

|  |
| --- |
| Neil Donnelly (10108823)  5/27/2013 |

Table of Contents

[1. Parent Widget 57](#_Toc357355637)

[1.1 Header File 57](#_Toc357355638)

[1.2 Class File 58](#_Toc357355639)

[2. Audio Manager 60](#_Toc357355640)

[2.1 Header File 60](#_Toc357355641)

[2.2 Class File 61](#_Toc357355642)

[3. Input Manager 68](#_Toc357355643)

[3.1 Header File 68](#_Toc357355644)

[3.2 Class File 68](#_Toc357355645)

[4. Animation Manager 71](#_Toc357355646)

[4.1 Header File 71](#_Toc357355647)

[4.2 Class File 71](#_Toc357355648)

# 1. Parent Widget

## 1.1 Header File

/\*

This parent widget extends from the Widget class but includes behaviour to update all the children

\*/

#ifndef PARENTWIDGET\_H

#define PARENTWIDGET\_H

#include "AudioManager.h"

#include "Widget.h"

#include <list>

namespace TDE

{

class ParentWidget : public Widget

{

public:

ParentWidget(int x, int y, int width, int height, ParentWidget\* parent);

~ParentWidget(void);

//Extends these functions as it will call each childs version of the //function as well

virtual void Update();

virtual void Draw(TDEGraphics\* g);

virtual void Hide();

virtual void Show();

//Adds and removes widgets from the parent's list of children

virtual void AddChild(Widget\* child);

virtual void RemoveChild(Widget\* child);

//Returns a keyboard and mouse subject to the child

//Each parent will check its parent until the subject is gotten from the //root widget

virtual KeySubject\* GetKeyboard();

virtual MouseSubject\* GetMouse();

virtual TDEGraphics\* GetGraphics() {return mParent->GetGraphics();};

virtual AudioManager\* GetAudioManager() {

return mParent->GetAudioManager();};

//Used by button widgets to let the parent know it was pressed

virtual void OnBtnClick(int btnID);

virtual void OnBtnRelease(int btnID);

protected:

//List of child widgets, the parent and the number of children

list<Widget\*> mChildren;

ParentWidget\* mParent;

int mNumChildren;

};

}

#endif

## 1.2 Class File

#include "ParentWidget.h"

namespace TDE

{

ParentWidget::ParentWidget(int x, int y, int width, int height,

ParentWidget\* parent)

: Widget(x,y,width,height, parent)

{

mNumChildren = 0;

mParent = parent;

}

ParentWidget::~ParentWidget(void)

{

}

void ParentWidget::Update()

{

//Iterates though the list of children and updates each one

if(!(mChildren.empty()))

{

for(list<TDE::Widget\*>::iterator it = mChildren.begin();

it != mChildren.end(); it++)

{

(\*it)->Update();

}

}

}

//Iterates through list of children and draws each

void ParentWidget::Draw(TDEGraphics\* g)

{

if(!(mChildren.empty()))

{

for(list<TDE::Widget\*>::iterator it = mChildren.begin();

it != mChildren.end(); it++)

{

if((\*it)->IsActive())

(\*it)->Draw(g);

}

}

}

//Hides all children

void ParentWidget::Hide()

{

if(!(mChildren.empty()))

{

for(list<TDE::Widget\*>::iterator it = mChildren.begin();

it != mChildren.end(); it++)

{

(\*it)->Hide();

}

}

}

//Shows all children

void ParentWidget::Show()

{

if(!(mChildren.empty()))

{

for(list<TDE::Widget\*>::iterator it = mChildren.begin();

it != mChildren.end(); it++)

{

(\*it)->Show();

}

}

}

//Adds child to list of children

void ParentWidget::AddChild(Widget\* child)

{

if(!child)

return;

for(list<TDE::Widget\*>::iterator it = mChildren.begin();

it != mChildren.end(); it++)

{

if(child == (\*it))

return;

}

child->setID(mNumChildren);

mChildren.push\_back(child);

mNumChildren++;

}

//Removes child from list of children using the pointer to compare

void ParentWidget::RemoveChild(Widget\* child)

{

if(!child)

return;

if(mChildren.empty())

return;

mChildren.remove(child);

mNumChildren--;

}

KeySubject\* ParentWidget::GetKeyboard()

{

return mNoParent ? NULL : mParent->GetKeyboard();

}

MouseSubject\* ParentWidget::GetMouse()

{

return mNoParent ? NULL : mParent->GetMouse();

}

void ParentWidget::OnBtnClick(int btnID)

{

return;

}

void ParentWidget::OnBtnRelease(int btnID)

{

return;

}

}

# 2. Audio Manager

## 2.1 Header File

#ifndef AUDIO\_MGR

#define AUDIO\_MGR

#define NUM\_CHANNELS 64

#include "TDE\_Music.h"

#include "TDE\_Sound.h"

#include <map>

#include <vector>

#include <queue>

#include <array>

namespace TDE

{

class AudioManager

{

public:

AudioManager(void);

~AudioManager(void);

bool Init();

void CleanUp();

void Update();

bool LoadSoundFile(std::string name, std::string path);

bool LoadMusicFile(std::string name, std::string path);

TDE\_Music\* GetCurrentMusic();

TDE\_Music\* GetMusic(std::string name);

bool PlayMusic(std::string name, int repeats);

bool PlayMusic(TDE\_Music\* m, int repeats);

bool FadeInMusic(std::string name, int repeats, int fadeTime);

bool FadeInMusic(TDE\_Music\* m, int repeats, int fadeTime);

void FadeOutMusic(int fadeTime);

void PauseMusic();

void ResumeMusic();

void StopMusic();

void SetMusicVolume(int vol);

void IncremenetMusicVolume();

void DecrementMusicVolume();

TDE\_Sound\* GetSound(std::string name);

bool PlaySound(std::string name, int repeats);

bool PlaySound(TDE\_Sound\* s, int repeats);

void PauseSound(TDE\_Sound\* s);

void PauseSound(std::string name);

void ResumeSound(TDE\_Sound\* s);

void ResumeSound(std::string name);

void StopSound(TDE\_Sound\* s);

void StopSound(std::string name);

void FreeSound(std::string name);

void ResumeAllSounds();

void PauseAllSounds();

void StopAllSounds();

void FreeAllSounds();

void SetVolumeForSounds(int v);

void IncrementSoundVolume();

void DecrementSoundVolume();

void ClearWaitingSounds();

void ChannelDone(int channel);

void FinishedChannel(int channel);

int GetNumChannels() {return mNumChannels;};

int GetNumFreeChannels() {return mAvailableChannels;};

int GetNumWaiting() {return mNumWaiting;};

int GetVolume() {return mChannelVolume;};

int GetMusVolume() {return Mix\_VolumeMusic(-1);};

int GetChannelsWaitingRefresh() {return mDoneChannels.size();};

private:

int FindChannel();

int mNumChannels;

int mAvailableChannels;

int mNumWaiting;

int mChannelVolume;

int mMusVolume;

TDE\_Music\* mMusic;

std::map<std::string, TDE\_Sound> mSoundMap;

std::map<std::string, TDE\_Music> mMusicMap;

std::array<TDE\_Sound\*, NUM\_CHANNELS> mChannels;

std::vector<int> mDoneChannels;

std::queue<std::pair<TDE\_Sound\*, int>> mWaitingSounds;

};

class ChannelHandler

{

public:

static void SetCallback(AudioManager \*am);

static void DoneChannel(int c);

private:

static AudioManager\* AudioMgr;

};

}

#endif

## 2.2 Class File

#include "AudioManager.h"

using namespace std;

namespace TDE

{

AudioManager::AudioManager(void)

{

mNumChannels = 0;

mAvailableChannels = 0;

mNumWaiting = 0;

mMusic = NULL;

mChannels.fill(NULL);

mChannelVolume = 64;

mMusVolume = 128;

}

AudioManager::~AudioManager(void)

{

}

bool AudioManager::Init()

{

if(SDL\_Init(SDL\_INIT\_AUDIO) == -1)

{

printf("SDL\_Init: %s\n", SDL\_GetError());

return false;

}

if(Mix\_OpenAudio(22050, MIX\_DEFAULT\_FORMAT, 2, 1024)==-1)

{

printf("Mix\_OpenAudio: %s\n", Mix\_GetError());

return false;

}

mNumChannels = mAvailableChannels = Mix\_AllocateChannels(NUM\_CHANNELS);

ChannelHandler::SetCallback(this);

Mix\_Volume(-1, mChannelVolume);

return true;

}

void AudioManager::CleanUp()

{

for(map<string, TDE\_Sound>::iterator it = mSoundMap.begin();

it != mSoundMap.end(); it++)

{

it->second.Delete();

}

for(map<string, TDE\_Music>::iterator it = mMusicMap.begin();

it != mMusicMap.end(); it++)

{

Mix\_FreeMusic(it->second.mMusic);

}

mMusic = NULL;

Mix\_CloseAudio();

Mix\_Quit();

}

void AudioManager::Update()

{

while(mDoneChannels.size() > 0)

{

ChannelDone(mDoneChannels.back());

mDoneChannels.pop\_back();

}

}

bool AudioManager::LoadSoundFile(string name, string path)

{

TDE\_Sound s(name, path);

if(s.ValidateSound())

{

mSoundMap.insert(pair<string, TDE\_Sound>(name, s));

return true;

}

else return false;

}

bool AudioManager::LoadMusicFile(string name, string path)

{

TDE\_Music m(name, path);

if(m.ValidateMusic())

{

mMusicMap.insert(pair<string, TDE\_Music>(name, m));

return true;

}

else return false;

}

TDE\_Music\* AudioManager::GetCurrentMusic()

{

return mMusic;

}

TDE\_Music\* AudioManager::GetMusic(string name)

{

map<string, TDE\_Music>::iterator it = mMusicMap.find(name);

if(it == mMusicMap.end())

return NULL;

else return &(it->second);

}

bool AudioManager::PlayMusic(string name, int repeats)

{

map<string, TDE\_Music>::iterator it = mMusicMap.find(name);

if(it == mMusicMap.end())

return NULL;

else

{

it->second.Play(repeats);

mMusic = &(it->second);

return true;

}

}

bool AudioManager::PlayMusic(TDE\_Music\* m, int repeats)

{

if(!(m->ValidateMusic()))

return false;

mMusic = m;

m->Play(repeats);

return true;

}

bool AudioManager::FadeInMusic(string name, int repeats, int fadeTime)

{

map<string, TDE\_Music>::iterator it = mMusicMap.find(name);

if(it == mMusicMap.end())

return false;

else

{

it->second.FadeIn(repeats, fadeTime);

mMusic = &(it->second);

return true;

}

}

bool AudioManager::FadeInMusic(TDE\_Music\* m, int repeats, int fadeTime)

{

if(!(m->ValidateMusic()))

return false;

mMusic = m;

m->FadeIn(repeats, fadeTime);

return true;

}

void AudioManager::FadeOutMusic(int fadeTime)

{

if(mMusic)

mMusic->FadeOut(fadeTime);

}

void AudioManager::PauseMusic()

{

if(mMusic)

mMusic->Pause();

}

void AudioManager::ResumeMusic()

{

if(mMusic)

mMusic->Resume();

}

void AudioManager::StopMusic()

{

if(mMusic)

mMusic->Stop();

}

void AudioManager::SetMusicVolume(int vol)

{

Mix\_VolumeMusic(vol);

mMusVolume = Mix\_VolumeMusic(-1);

}

void AudioManager::IncremenetMusicVolume()

{

SetMusicVolume(mMusVolume+1);

}

void AudioManager::DecrementMusicVolume()

{

SetMusicVolume(mMusVolume-1);

}

TDE\_Sound\* AudioManager::GetSound(string name)

{

map<string, TDE\_Sound>::iterator it = mSoundMap.find(name);

if(it == mSoundMap.end())

return NULL;

else return &(it->second);

}

bool AudioManager::PlaySound(std::string name, int repeats)

{

TDE\_Sound\* s = GetSound(name);

if(!s)

return false;

else return PlaySound(s, repeats);

}

bool AudioManager::PlaySound(TDE\_Sound\* s, int repeats)

{

if(!(s->ValidateSound()))

return false;

int channel = FindChannel();

if(channel < 0)

{

s->SetWaiting(true);

if(mNumWaiting <= 32)

{

mWaitingSounds.push(pair<TDE\_Sound\*, int>(s, repeats));

mNumWaiting++;

}

else return false;

}

else

{

s->Play(channel, repeats);

mChannels[channel] = s;

}

return true;

}

void AudioManager::ResumeSound(TDE\_Sound\* s)

{

if(s->IsPaused())

s->Resume();

}

void AudioManager::ResumeSound(std::string name)

{

TDE\_Sound\* s = GetSound(name);

if(s) ResumeSound(s);

}

void AudioManager::PauseSound(TDE\_Sound\* s)

{

if(s->IsPlaying())

s->Pause();

}

void AudioManager::PauseSound(std::string name)

{

TDE\_Sound\* s = GetSound(name);

if(s) PauseSound(s);

}

void AudioManager::StopSound(TDE\_Sound\* s)

{

s->Stop();

}

void AudioManager::StopSound(std::string name)

{

TDE\_Sound\* s = GetSound(name);

if(s) StopSound(s);

}

void AudioManager::FreeSound(std::string name)

{

TDE\_Sound\* s = GetSound(name);

if(s) s->Delete();

}

void AudioManager::PauseAllSounds()

{

Mix\_Pause(-1);

}

void AudioManager::ResumeAllSounds()

{

Mix\_Resume(-1);

}

void AudioManager::StopAllSounds()

{

Mix\_HaltChannel(-1);

mChannels.fill(NULL);

}

void AudioManager::FreeAllSounds()

{

for(map<string, TDE\_Sound>::iterator it = mSoundMap.begin();

it != mSoundMap.end(); it++)

{

it->second.Delete();

}

mChannels.fill(NULL);

}

void AudioManager::SetVolumeForSounds(int v)

{

Mix\_Volume(-1, v);

mChannelVolume = Mix\_Volume(-1,-1);

}

void AudioManager::IncrementSoundVolume()

{

mChannelVolume = mChannelVolume == 128 ? 128 : mChannelVolume+1;

Mix\_Volume(-1, mChannelVolume);

}

void AudioManager::DecrementSoundVolume()

{

mChannelVolume = mChannelVolume == 0 ? 0 : mChannelVolume-1;

Mix\_Volume(-1, mChannelVolume);

}

void AudioManager::ClearWaitingSounds()

{

while(!(mWaitingSounds.empty()))

mWaitingSounds.pop();

}

int AudioManager::FindChannel()

{

for(int i = 0; i < NUM\_CHANNELS; i++)

{

if(!(mChannels[i]))

return i;

}

return -1;

}

void AudioManager::ChannelDone(int channel)

{

if(mNumWaiting == 0)

{

if(mChannels[channel])

mChannels[channel]->Stop();

mChannels[channel] = NULL;

}

else

{

pair<TDE\_Sound\*, int> p = mWaitingSounds.front();

TDE\_Sound\* s = p.first;

int repeats = p.second;

mChannels[channel] = s;

s->Play(channel, repeats);

mWaitingSounds.pop();

mNumWaiting--;

}

}

void AudioManager::FinishedChannel(int channel)

{

mDoneChannels.push\_back(channel);

}

AudioManager\* ChannelHandler::AudioMgr;

void ChannelHandler::SetCallback(AudioManager \*am)

{

AudioMgr = am;

Mix\_ChannelFinished(DoneChannel);

}

void ChannelHandler::DoneChannel(int channel)

{

AudioMgr->FinishedChannel(channel);

}

}

# 3. Input Manager

## 3.1 Header File

/\*

The input manager is in control of collecting the input from SDL and supplying it to

the subjects in the observer pattern to notify their subscribers of the change

\*/

#ifndef INPUT\_MGR\_H

#define INPUT\_MGR\_H

#include <vector>

#include "Includes.h"

#include "InputSubject.h"

namespace TDE

{

class InputManager

{

public:

InputManager();

~InputManager();

//Input manager checks for any new input

bool Update();

//Returns a pointer to the subjects in the observer pattern

//Used by the observers to subscribe

MouseSubject\* GetMouseSubject();

KeySubject\* GetKeySubject();

private:

//The subjects for the input

MouseSubject mMouseSubject;

KeySubject mKeySubject;

//Mouse State is a struct containing the latest info on the mouse

//i.e. positition and state of the buttons

MouseState mMouseState;

};

}

#endif

## 3.2 Class File

#include "InputManager.h"

namespace TDE

{

InputManager::InputManager()

{

//Creates objects for the mouse and keyboard subjects

mMouseSubject = MouseSubject();

mKeySubject = KeySubject();

//Initialises the mouse state

mMouseState.x = 0;

mMouseState.y = 0;

mMouseState.leftClicked = false;

mMouseState.rightClicked = false;

mMouseState.middleClicked = false;

}

InputManager::~InputManager()

{

}

bool InputManager::Update()

{

//Polls the SDL event handler and for each key press it detected decides //what to do with it

SDL\_Event aEvent;

while(SDL\_PollEvent(&aEvent))

{

switch(aEvent.type)

{

//For any key press or release, let the keyboard subject know

case SDL\_KEYUP:

case SDL\_KEYDOWN:

mKeySubject.Notify(&aEvent.key);

break;

//Updates the mouse state with the new position and notifies the //Mouse subject

case SDL\_MOUSEMOTION:

mMouseState.x = aEvent.motion.x;

mMouseState.y = aEvent.motion.y;

mMouseSubject.Notify(mMouseState);

break;

//If a button is pressed, record what buttons are pressed and //notify the mouse subject

case SDL\_MOUSEBUTTONUP:

case SDL\_MOUSEBUTTONDOWN:

switch(aEvent.button.button)

{

case SDL\_BUTTON\_LEFT:

mMouseState.leftClicked = aEvent.button.state == SDL\_PRESSED ? true : false;

break;

case SDL\_BUTTON\_RIGHT:

mMouseState.rightClicked = aEvent.button.state == SDL\_PRESSED ? true : false;

break;

case SDL\_BUTTON\_MIDDLE:

mMouseState.middleClicked = aEvent.button.state == SDL\_PRESSED ? true : false;

break;

default:

break;

}

mMouseSubject.Notify(mMouseState);

break;

//Detects if the window is being closed, if so print it to the //output (for what its worth) and shut down

case SDL\_QUIT:

printf("Quitting\n");

exit(1);

return true;

break;

default:

break;

}

}

return false;

}

MouseSubject\* InputManager::GetMouseSubject()

{

return &mMouseSubject;

}

KeySubject\* InputManager::GetKeySubject()

{

return &mKeySubject;

}

}

# 4. Animation Manager

## 4.1 Header File

#ifndef ANIM\_MGR

#define ANIM\_MGR

#include "TDE\_Animation.h"

namespace TDE

{

class AnimationManager

{

public:

AnimationManager(TextureManager\* aTexMgr);

AnimationManager(void);

~AnimationManager(void);

bool LoadAnimation(std::string name, TDEImage\* cells[], int numCells);

bool LoadAnimation(std::string name, TDEImage\* anIm, int cellWidth,

int cellHeight, int numCells);

bool LoadAnimation(std::string name, std::string path, int cellWidth,

int cellHeight, int numCells);

bool LoadAnimation(std::string name, std::string path, int cellWidth,

int cellHeight, int totalWidth, int totalHeight, int numCells);

TDE\_Animation GetAnim(std::string name);

bool RegisterAnim(TDE\_Animation\* anim);

bool DeregisterAnim(TDE\_Animation\* anim);

void UpdateAll();

void PauseAll();

void ResumeAll();

void StopAll();

void DeleteAll();

int GetNumStoredAnims();

int GetRegisteredAnims();

private:

TextureManager\* mTexMgr;

std::map<std::string, TDE\_Animation> mAnimMap;

std::vector<TDE\_Animation\*> mControlVec;

int mStoredAnims;

int mAnimsToControl;

};

}

#endif

## 4.2 Class File

#include "AnimationManager.h"

using namespace std;

namespace TDE

{

AnimationManager::AnimationManager(TextureManager\* aTexMgr)

{

mTexMgr = aTexMgr;

mAnimMap.clear();

mControlVec.clear();

}

AnimationManager::AnimationManager(void)

{

mTexMgr = NULL;

}

AnimationManager::~AnimationManager(void)

{

}

bool AnimationManager::LoadAnimation(string name, TDEImage\* cells[],

int numCells)

{

TDE\_Animation anim = TDE\_Animation(name, cells, numCells);

if(anim.GetNumCells() > 0)

{

mAnimMap.insert(pair<string, TDE\_Animation>(name, anim));

return true;

}

else return false;

}

bool AnimationManager::LoadAnimation(string name, TDEImage\* anIm,

int cellWidth, int cellHeight, int numCells)

{

TDE\_Animation anim = TDE\_Animation(TDE\_Animation(name, anIm,

mTexMgr->GetTexture(anIm->GetTexRef()),

cellWidth, cellHeight, numCells));

if(anim.GetNumCells() > 0)

{

mAnimMap.insert(pair<string, TDE\_Animation>(name, anim));

return true;

}

else return false;

}

bool AnimationManager::LoadAnimation(string name, string path, int cellWidth,

int cellHeight, int numCells)

{

if(!(mTexMgr))

return false;

if(!(mTexMgr->LoadImage(path.c\_str(), name.c\_str())))

{

return false;

}

TDEImage\* anIm = mTexMgr->GetImage(name);

if(anIm)

{

TDE\_Animation anim = TDE\_Animation(TDE\_Animation(name, anIm,

mTexMgr->GetTexture(anIm->GetTexRef()), cellWidth,

cellHeight, numCells));

if(anim.GetNumCells() > 0)

{

mAnimMap.insert(pair<string, TDE\_Animation>(name, anim));

return true;

}

}

return false;

}

bool AnimationManager::LoadAnimation(string name, string path, int cellWidth,

int cellHeight, int totalWidth, int totalHeight, int numCells)

{

if(!(mTexMgr))

return false;

if(!(mTexMgr->LoadImage(path.c\_str(), name.c\_str(), totalWidth,

totalHeight)))

{

return false;

}

TDEImage\* anIm = mTexMgr->GetImage(name);

if(anIm)

{

TDE\_Animation anim = TDE\_Animation(TDE\_Animation(name, anIm,

mTexMgr->GetTexture(anIm->GetTexRef()), cellWidth,

cellHeight, numCells));

if(anim.GetNumCells() > 0)

{

mAnimMap.insert(pair<string, TDE\_Animation>(name, anim));

return true;

}

}

return false;

}

TDE\_Animation AnimationManager::GetAnim(string name)

{

map<string, TDE\_Animation>::iterator it = mAnimMap.find(name);

if(it == mAnimMap.end())

return TDE\_Animation();

else return (it->second);

}

bool AnimationManager::RegisterAnim(TDE\_Animation\* anim)

{

if(anim)

{

mControlVec.push\_back(anim);

return true;

}

return false;

}

bool AnimationManager::DeregisterAnim(TDE\_Animation\* anim)

{

vector<TDE\_Animation\*>::iterator it;

for(it = mControlVec.begin(); it != mControlVec.end(); it++)

{

if((\*it) == anim)

{

mControlVec.erase(it);

return true;

}

}

return false;

}

void AnimationManager::UpdateAll()

{

for(int i = 0; i < mControlVec.size(); i++)

{

mControlVec[i]->Update();

}

}

void AnimationManager::PauseAll()

{

for(int i = 0; i < mControlVec.size(); i++)

{

mControlVec[i]->Pause();

}

}

void AnimationManager::ResumeAll()

{

for(int i = 0; i < mControlVec.size(); i++)

{

mControlVec[i]->Resume();

}

}

void AnimationManager::StopAll()

{

for(int i = 0; i < mControlVec.size(); i++)

{

mControlVec[i]->Stop();

}

}

void AnimationManager::DeleteAll()

{

for(int i = 0; i < mControlVec.size(); i++)

{

mControlVec[i]->Delete();

}

mAnimMap.clear();

}

int AnimationManager::GetNumStoredAnims()

{

return mAnimMap.size();

}

int AnimationManager::GetRegisteredAnims()

{

return mControlVec.size();

}

}