#include <SDL.h>

#include <SDL\_image.h>

#include <stdio.h>

#include <string>

#include<iostream>

//Screen dimension constants

const int SCREEN\_WIDTH = 942;

const int SCREEN\_HEIGHT = 656;

struct plate{

int g;//gem number

int x;

int y;

plate(){}

};

//Texture wrapper class

class LTexture

{

public:

//Initializes variables

LTexture();

//Deallocates memory

~LTexture();

//Loads image at specified path

bool loadFromFile( std::string path );

//Deallocates texture

void free();

//Renders texture at given point

void render( int x, int y );

//Gets image dimensions

int getWidth();

int getHeight();

private:

//The actual hardware texture

SDL\_Texture\* mTexture;

//Image dimensions

int mWidth;

int mHeight;

};

void getboard();

//Starts up SDL and creates window

bool init();

//Loads media

bool loadMedia();

//Frees media and shuts down SDL

void close();

//The window we'll be rendering to

SDL\_Window\* gWindow = NULL;

//The window renderer

SDL\_Renderer\* gRenderer = NULL;

LTexture gBackgroundTexture;

LTexture gem[5];

LTexture blank;

LTexture cursor;

LTexture::LTexture()

{

//Initialize

mTexture = NULL;

mWidth = 0;

mHeight = 0;

}

LTexture::~LTexture()

{

//Deallocate

free();

}

bool LTexture::loadFromFile( std::string path )

{

//Get rid of preexisting texture

free();

//The final texture

SDL\_Texture\* newTexture = NULL;

//Load image at specified path

SDL\_Surface\* loadedSurface = IMG\_Load( path.c\_str() );

if( loadedSurface == NULL )

{

printf( "Unable to load image %s! SDL\_image Error: %s\n", path.c\_str(), IMG\_GetError() );

}

else

{

//Color key image

SDL\_SetColorKey( loadedSurface, SDL\_TRUE, SDL\_MapRGB( loadedSurface->format, 0xFF, 0xFF, 0xFF ) );

//Create texture from surface pixels

newTexture = SDL\_CreateTextureFromSurface( gRenderer, loadedSurface );

if( newTexture == NULL )

{

printf( "Unable to create texture from %s! SDL Error: %s\n", path.c\_str(), SDL\_GetError() );

}

else

{

//Get image dimensions

mWidth = loadedSurface->w;

mHeight = loadedSurface->h;

}

//Get rid of old loaded surface

SDL\_FreeSurface( loadedSurface );

}

//Return success

mTexture = newTexture;

return mTexture != NULL;

}

void LTexture::free()

{

//Free texture if it exists

if( mTexture != NULL )

{

SDL\_DestroyTexture( mTexture );

mTexture=NULL;

}

}

void LTexture::render( int x, int y )

{

//Set rendering space and render to screen

SDL\_Rect renderQuad = { x, y, mWidth, mHeight };

SDL\_RenderCopy( gRenderer, mTexture, NULL, &renderQuad );

}

int LTexture::getWidth()

{

return mWidth;

}

int LTexture::getHeight()

{

return mHeight;

}

bool init()

{

//Initialization flag

bool success = true;

//Initialize SDL

if( SDL\_Init( SDL\_INIT\_VIDEO ) < 0 )

{

printf( "SDL could not initialize! SDL Error: %s\n", SDL\_GetError() );

success = false;

}

else

{

//Set texture filtering to linear

if( !SDL\_SetHint( SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1" ) )

{

printf( "Warning: Linear texture filtering not enabled!" );

}

//Create window

gWindow = SDL\_CreateWindow( "SDL Tutorial", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, SCREEN\_WIDTH, SCREEN\_HEIGHT, SDL\_WINDOW\_SHOWN );

if( gWindow == NULL )

{

printf( "Window could not be created! SDL Error: %s\n", SDL\_GetError() );

success = false;

}

else

{

//Create renderer for window

gRenderer = SDL\_CreateRenderer( gWindow, -1, SDL\_RENDERER\_ACCELERATED );

if( gRenderer == NULL )

{

printf( "Renderer could not be created! SDL Error: %s\n", SDL\_GetError() );

success = false;

}

else

{

//Initialize renderer color

SDL\_SetRenderDrawColor( gRenderer, 0xFF, 0xFF, 0xFF, 0xFF );

//Initialize PNG loading

int imgFlags = IMG\_INIT\_PNG;

if( !( IMG\_Init( imgFlags ) & imgFlags ) )

{

printf( "SDL\_image could not initialize! SDL\_image Error: %s\n", IMG\_GetError() );

success = false;

}

}

}

}

return success;

}

bool loadMedia()

{

//Loading success flag

bool success = true;

blank.loadFromFile("image/blank.png");

cursor.loadFromFile( "image/cursor.png" );

gem[1].loadFromFile( "image/1.png" );

gem[2].loadFromFile( "image/2.png" );

gem[3].loadFromFile( "image/3.png" );

gem[4].loadFromFile( "image/4.png" );

gem[5].loadFromFile( "image/5.png" );

if( !gBackgroundTexture.loadFromFile( "image/background (2).png" ) )

{

printf( "Failed to load background texture image!\n" );

success = false;

}

return success;

}

void close()

{

//Free loaded images

for(int i=1;i<6;i++){

gem[i].free();

}

gBackgroundTexture.free();

cursor.free();

//Destroy window

SDL\_DestroyRenderer( gRenderer );

SDL\_DestroyWindow( gWindow );

gWindow = NULL;

gRenderer = NULL;

//Quit SDL subsystems

IMG\_Quit();

SDL\_Quit();

}

void getboard(plate p[12][12]){

int b=40;int a;int r;

for(int i=1;i<=10;i++){

a=68;

for(int j=1;j<=10;j++){

r=rand() % 5 + 1;

p[i][j].g=r;

p[i][j].x=a;a+=59;

p[i][j].y=b;

}

b+=59;

}

}

int main( int argc, char\* args[] )

{

if( !init() )

{

printf( "Failed to initialize!\n" );

}

else

{

//Load media

if( !loadMedia() )

{

printf( "Failed to load media!\n" );

}

else

{

plate p[12][12];

for(int i=0;i<7;i++){getboard(p);}

//Main loop flag

bool quit = false;

//Event handler

SDL\_Event e;

int x=68,y=40,a=1,b=1;

//While application is running

while( !quit )

{

//Handle events on queue

while( SDL\_PollEvent( &e ) != 0 )

{

//User requests quit

if( e.type == SDL\_QUIT )

{

quit = true;

}

else if( e.type == SDL\_KEYDOWN )

{

//Select surfaces based on key press

switch( e.key.keysym.sym )

{

case SDLK\_UP:

if(y==40){y=571;a=10;break;}

y-=59;a--;

break;

case SDLK\_DOWN:

if(y==571){y=40;a=1;break;}

y+=59;a++;

break;

case SDLK\_LEFT:

if(x==68){x=599;b=10;break;}

x-=59;b--;

break;

case SDLK\_RIGHT:

if(x==599){x=68;b=1;break;}

x+=59;b++;

break;

default:

break;

}

}

}

std::cout<<'t'<<a<<" "<<b<<" ";

//Clear screen

SDL\_SetRenderDrawColor( gRenderer, 0xFF, 0xFF, 0xFF, 0xFF );

SDL\_RenderClear( gRenderer );

//Render background texture to screen

gBackgroundTexture.render( 0, 0 );

for(int i=1;i<=10;i++){

for(int j=1;j<=10;j++){

gem[p[i][j].g].render(p[i][j].x,p[i][j].y);

}}

cursor.render(x,y);

//Update screen

SDL\_RenderPresent( gRenderer );

}

}

}

//Free resources and close SDL

close();

return 0;

}