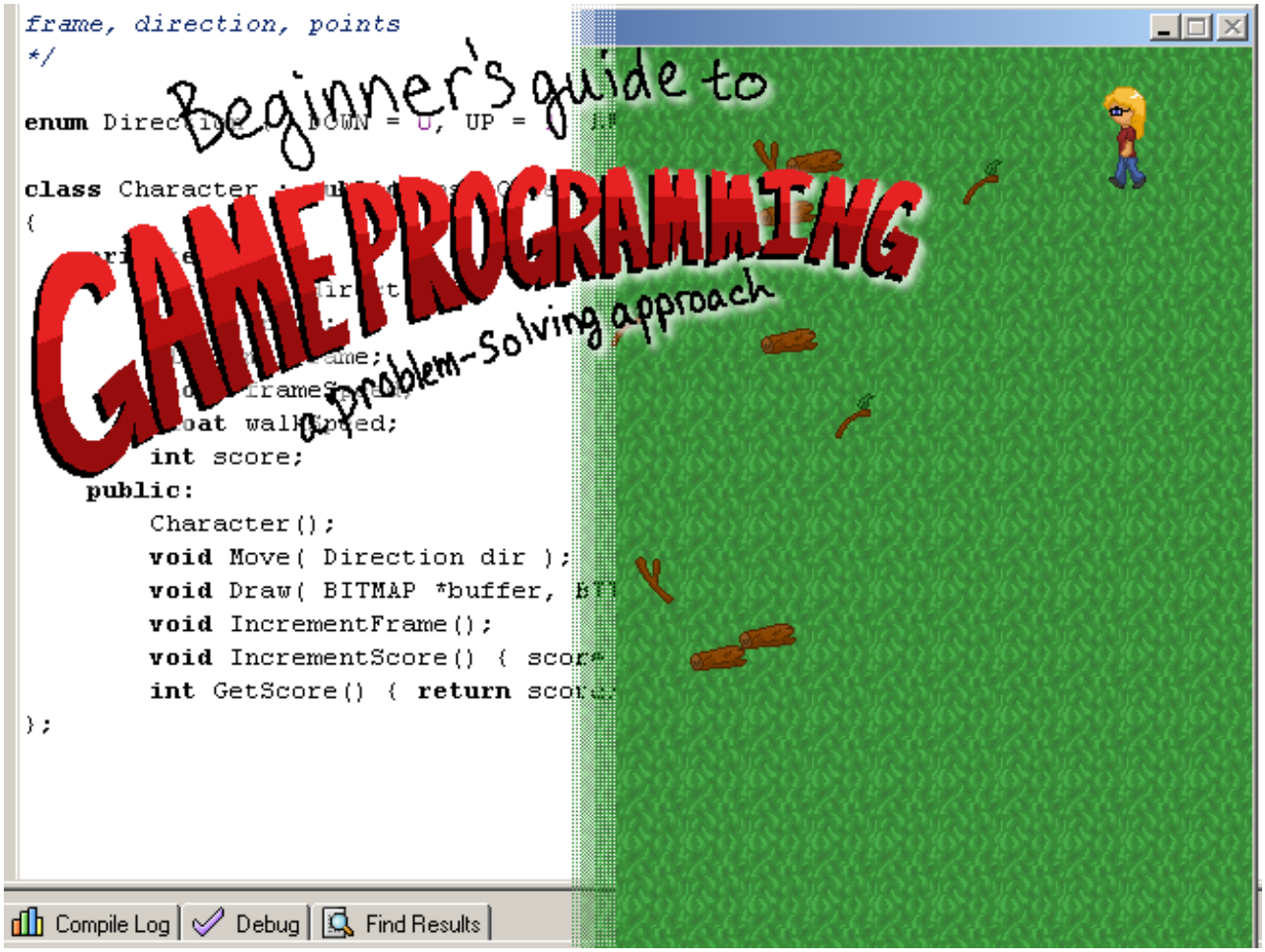


Episode 1 v0.2
Writing your own Pickin' Sticks with C++ and Allegro or SDL

(aka a game with basic movement and collision)

Tutorial, game, and graphics by Rachel J. Morris, May - July 2009

To see a video version of this tutorial, other tutorials, and check out my open-source C++ games, please go to www.moosader.com

Table of Contents

Revision History

July 25, 2009
Fixed up code in SDL's Graphics and Sound portion

May ?, 2009
Initial writing

Introduction

Welcome to Beginner's Guide to Game Programming – A problem-solving approach. This tutorial is designed to teach you how to write a super-basic game in C++ using either Allegro or SDL to handle graphics, sound, input, and timers.

First off, **yes that is a lot of pages!** But I've made liberal use of whitespace to try to keep the document easy to read, plus there is a lot to cover if this is your first game! Please keep an open mind and give it a try. Everything is pretty basic, and if you don't understand something I am available for any questions via either my YouTube channel (http://www.youtube.com/LusikkaMage) or my email address (RachelJMorris@gmail.com).

This tutorial will cover the key elements of making the sample game, but will not actually give you the full code on how to implement it. (Thus, the “problem-solving” approach).

For this first tutorial, we will be making a game called Pickin' Sticks (though you can use whatever graphics and sounds you want in your version), and while the source code is online, I suggest you try to figure out how to write everything on your own first. This way, you begin to build the skills you need to write more complicated games.

There will also be a video tutorial portion of the tutorial, covering the same things in relatively small video chunks. Major questions answered by viewers (and readers) will be updated both in the text format (this thing), and as additional videos.

Also, a lot of the code examples in this tutorial are bare-bones. It is up to you to figure out how you want to implement them in a class, and I will try to have some example classes later on in the chapters.

What are we making?

Pickin' Sticks is a very basic game I use as a sort of “Hello World” game I make when trying something new – a new library, language, etc. The gist of the game is that you have a top-down view of your character and a single stick on the screen, which is placed randomly. When you touch the stick, a point is added and the stick gets new coordinates, which are also randomly generated.

Behold, mortals.

What you should already know

This tutorial assumes that you have no prior knowledge of Allegro or SDL, but are familiar enough with C++. This includes the following:

- Variables
- If Statements and Loops
- Functions
- Classes and Inheritance
- Pointers (primarily for passing to a function, so it can change more than one value)

Setting up Allegro or SDL

It is up to you which library you choose to use for your projects. Allegro is a bit easier than SDL, but SDL has more functionality, such as Multithreading and even Socket support. Both can be used with OpenGL, though SDL is more widely used than Allegro. Maybe a not-so-important concern is that people will take you less seriously if you use Allegro (this is just my own personal experience). If I end up writing as many tutorials as I would like, I will probably eventually change to SDL only, but for the basics I'll have Allegro code too.

Also, which IDE you choose to use is up to you, but I only have experience using Visual Studio with SDL + OpenGL. For the most part, I use Code::Blocks as it is free. If you've never installed a library for your IDE before, you may look int DevC++, which uses things called “DevPak”s that auto-install libraries for you just by double-clicking them.

As a note, this tutorial uses Allegro 4.2. There is an Allegro 4.9 out, which is the beta for Allegro 5, and some things have changed with it.

Free IDE download links

Visual C++ Express:	http://www.microsoft.com/Express/vc/
DevC++:	http://www.bloodshed.net/devcpp.html
Code::Blocks:	http://www.codeblocks.org/

Tutorials for setting up Allegro

Visual C++:	http://lmgtfy.com/?q=Set+up+Allegro+Visual+C%2B%2B
DevC++:	http://lmgtfy.com/?q=Set+up+Allegro+DevC%2B%2B
Code::Blocks:	http://lmgtfy.com/?q=Set+up+Allegro+Code%3A%3ABlocks

Tutorials for setting up SDL

LazyFoo's Site:	http://lazyfoo.net/SDL_tutorials/lesson01/index.php

Other handy tutorials and resources

Allegro Tutorials

Loomsoft Games and Tutorials – *Covers the super-basics of Allegro*
http://www.loomsoft.net/resources/alltut/alltut_index.htm

Allegro Quick Reference – *Immensely helpful resource. Not really a tutorial, but gives clear explanations of different Allegro functions.*
http://www.connellybarnes.com/documents/quick_reference.html

SDL Tutorials

Lazy Foo's Tutorials – *Pretty much the one-stop location for SDL tutorials*
http://lazyfoo.net/SDL_tutorials/index.php

In general

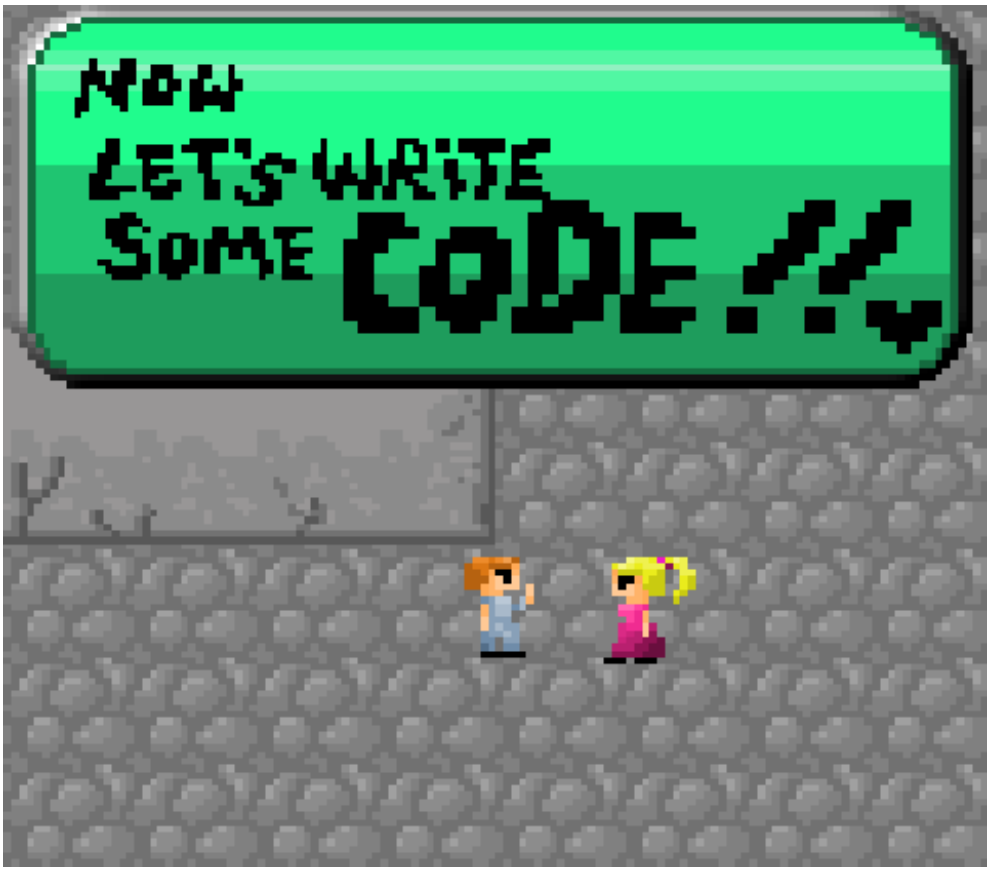
Falco's "Where to Begin" videos
http://www.youtube.com/view_play_list?p=93FE4FF8C4CB5498

GameDev's beginner section – *These are pretty general, and kind of old.*
http://www.gamedev.net/reference/start_here/

Resources for public domain graphics and sounds

When creating games, you should try to get used to using original or public-domain resources, rather than, say, sprites ripped from another game. This way, there are no concerns with Intellectual Property infringement.

On my site, http://www.moosader.com , there is a Resources section that has a few public domain songs and sprites that you can use for any projects. You can also find public domain resources by googling "public domain graphics/sound".


NOW
LET'S WRITE
SOME CODE!!

Lesson 1: Setting up a window

Before you can have fancy graphics and a sweet game, you first need to display a window on the screen!

Allegro initialization:

When you create a new project, make a blank console application. You can get to the linker by Project > Compiler options in Code::Blocks or Project > Project Options in DevC++.

In the linker, you'll write

-lalleg

And that's it!

```
#include <allegro.h>

int main()
{
    /* Initialization */
    allegro_init();
    install_keyboard();
    install_timer();
    install_mouse();
    install_sound( DIGI_AUTODETECT, MIDI_AUTODETECT, 0 );
    set_color_depth( 16 );

    bool fullscreen = false;

    if ( fullscreen == true )   // For fullscreen
        set_gfx_mode(GFX_AUTODETECT, 640, 480, 0, 0);
    else                        // For windowed
        set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

    BITMAP *buffer = create_bitmap( 640, 480 );
    /* End Initialization */

    /* Game loop and such would go here */
    while ( !key[KEY_ESC] )
    {
    /* Draw functions */
    blit( buffer, screen, 0, 0, 0, 0, 640, 480 );
    clear_bitmap( buffer );
    }

    /* Free memory afterwards! */
    destroy_bitmap( buffer );

    return 0;
}
END_OF_MAIN();
```

Right now, this will merely pop up a window and immediately close it, as there is nothing to keep the program running.

Make sure that there is END_OF_MAIN(); is at the end of your program or you'll get a "[Linker error] undefined reference to 'WinMain@16'" error.

SDL initialization:

When you create a new project, make a blank console application. You can get to the linker by Project > Compiler options in Code::Blocks or Project > Project Options in DevC++.

In the linker, you'll write

-lmingw32 -lSDLmain -lSDL -lSDL_mixer

```
#include "SDL/SDL.h"

int main( int argc, char *args[] )
{
    /* Initialization */
    SDL_Init( SDL_INIT_EVERYTHING );
    SDL_Surface *buffer;

    bool fullscreen = false;

    if ( fullscreen == true )   // For fullscreen
        buffer = SDL_SetVideoMode( 640, 480, 32, SDL_SWSURFACE |
            SDL_FULLSCREEN );
    else                        // For windowed
        buffer = SDL_SetVideoMode( 640, 480, 32, SDL_SWSURFACE );

    Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 ); // init sound

    // Set window caption
    SDL_WM_SetCaption( "Yay!", NULL );
    /* End Initialization */

    /* Game loop and such would go here */

    /* Close SDL.  Normally, you'd free surfaces, but since
      SDL_SetVideoMode was used on buffer, SDL_Quit knows to
      automatically free that for us. */
    SDL_Quit();

    return 0;
}
```

This will create a window but then immediately close it, as we do not have a game loop yet. For SDL, you need to have `int argc, char *args[]` in your main parameter list.

Also, before you can use Mix_OpenAudio to initialize sound, you have to set up an additional library in SDL.

Check out LazyFoo's **Setting up SDL Extension Libraries** tutorial:
http://lazyfoo.net/SDL_tutorials/lesson03/index.php
and get SDL_mixer files from:
http://www.libsdl.org/projects/SDL_mixer/

Double Buffering

Normally with a game, you will use ***double buffering***. This will keep the screen from flickering. Basically, it will draw everything to a surface (surface for SDL, bitmap for allegro), and once everything's all drawn to this surface, it is drawn to the screen all at once.

Cleaner code

The initialization functions should not be put in main. Normally, I would create a "Game" or "System" class, and put these into the constructor or a Setup function. That way, you will just have something like this:

```
int main()
{
    System myAllegroGame;

    return 0;
}
END_OF_MAIN();
```

One of your goals in writing games is to keep main as small as possible, and try to find a balance between Object Oriented code and using more C-style techniques where appropriate. While our computers don't really *need* optimized code, you *can* go overboard with OO techniques.

Make sure your exe works right!

Make sure you keep alleg42.dll in your game's folder (same directory as the .exe) for Allegro, and SDL.dll for an SDL project.

Lesson 2: Keyboard input

Allegro code:

`key[KEY_<name>]`

For example:

```
int main()
{
    System myAllegroGame;
    Player player;
    bool done = false;
    BITMAP *buffer = create_bitmap( 640, 480 );

    while ( !done )
    {
        /* Input */
        if ( key[KEY_ESC] )
            done = true;
        if ( key[KEY_F5] )
            myAllegroGame.ToggleFullscreen();

        if ( key[KEY_UP] )
            player.Move( UP );
        else if ( key[KEY_DOWN] )
            player.Move( DOWN );
        if ( key[KEY_LEFT] )
            player.Move( LEFT );
        else if ( key[KEY_RIGHT] )
            player.Move( RIGHT );

        /* Draw functions */
        blit( buffer, screen, 0, 0, 0, 0, 640, 480 );
        release_screen();
        clear_bitmap( buffer );
    }

    /* Free up memory or ELSE!! */
    destroy_bitmap( buffer );

    return 0;
}
END_OF_MAIN();
```

Small note:

if you have `else if( key[KEY_LEFT] )` instead of just `if( key[KEY_LEFT] )`, then it will make it so you cannot move diagonally. The way I have it above, the player can move vertically AND horizontally in the same game cycle.

Allegro keys

From the Allegro Documentation here:
http://alleg.sourceforge.net/latestdocs/en/alleg006.html

```
KEY_A ... KEY_Z,
KEY_0 ... KEY_9,
KEY_0_PAD ... KEY_9_PAD,
KEY_F1 ... KEY_F12,

KEY_ESC, KEY_TILDE, KEY_MINUS, KEY_EQUALS,
KEY_BACKSPACE, KEY_TAB, KEY_OPENBRACE, KEY_CLOSEBRACE,
KEY_ENTER, KEY_COLON, KEY_QUOTE, KEY_BACKSLASH,
KEY_BACKSLASH2, KEY_COMMA, KEY_STOP, KEY_SLASH,
KEY_SPACE,

KEY_INSERT, KEY_DEL, KEY_HOME, KEY_END, KEY_PGUP,
KEY_PGDN, KEY_LEFT, KEY_RIGHT, KEY_UP, KEY_DOWN,

KEY_SLASH_PAD, KEY_ASTERISK, KEY_MINUS_PAD,
KEY_PLUS_PAD, KEY_DEL_PAD, KEY_ENTER_PAD,

KEY_PRTSCR, KEY_PAUSE,

KEY_ABNT_C1, KEY_YEN, KEY_KANA, KEY_CONVERT, KEY_NOCONVERT,
KEY_AT, KEY_CIRCUMFLEX, KEY_COLON2, KEY_KANJI,

KEY_LSHIFT, KEY_RSHIFT,
KEY_LCONTROL, KEY_RCONTROL,
KEY_ALT, KEY_ALTGR,
KEY_LWIN, KEY_RWIN, KEY_MENU,
KEY_SCRLOCK, KEY_NUMLOCK, KEY_CAPSLOCK

KEY_EQUALS_PAD, KEY_BACKQUOTE, KEY_SEMICOLON, KEY_COMMAND
```

SDL code:

```
Uint8 key = SDL_GetKeyState( NULL );
if ( key[SDLK_DOWN] ) { ... }
```

For example:

```
Uint8 key*;
while ( !done )
{
      while ( SDL_PollEvent ( &event ) )
      {
            if ( event.type == SDL_QUIT )
            {
                  // if "X" button on title bar is hit
                  game.Done( true );
            }
      }

      // Get keyboard input
      key = SDL_GetKeyState( NULL );

      if ( key[SDLK_DOWN] )
            player.Move( DOWN );
}
```

SDL keys:

You can find a list of SDL keys in the SDL Documentation:
http://www.libsdl.org/cgi/docwiki.cgi/SDLKey
Here are the main ones:

```
SDLK_BACKSPACE    '\b'        backspace
SDLK_TAB          '\t'        tab
SDLK_RETURN       '\r'        return
SDLK_ESCAPE       '^['        escape
SDLK_SPACE        ' '         space
SDLK_PLUS         '+'         plus sign
SDLK_MINUS        '-'         minus sign
SDLK_a            'a'         a                 Same for other letters
SDLK_KP0                      keypad 0          Same for other numbers
SDLK_UP                       up arrow
SDLK_DOWN                     down arrow
```

SDLK_RIGHT	right arrow	
SDLK_LEFT	left arrow	
SDLK_F1	F1	*Same for other F keys*

Lesson 3: Displaying sprites

Now to actually display graphics!

The main types of graphics you'll have to deal with are animated (character sprites), and similarly still images but are in a tile-strip/sprite-sheet format like animated sprites (grid-based tiles for a map), and then a still picture, where you would output the entire image without having to crop any part of it (title screen).

Animated character sprite

Tileset

Still, full image (title screen)

One thing Allegro does for you is that it automatically makes that pink color (R 255, G 0, B 255 or #FF00FF) invisible. With SDL you'll have to define what color on your own.

What is "blitting"?

Refers to two bitmaps being combined, such as drawing all the on-screen game images to a buffer, and then drawing that to the screen.

Sprite dimension standards

Sprites should always have a power-of-two dimension. Allegro can load sprites at irregular sizes, but only because Allegro does work for you to resize the image file. Computer hardware reads image files at 2^x dimensions.

This might create unwanted extra space on your sprite, but you can make up for it in the code by specifying what region will be “solid” for collisions.

Allegro code

Allegro does have a **draw_bitmap** function, but I would shy away from using it, as you will almost always want to use **masked_blit** instead. draw_bitmap doesn't allow you to crop a portion of the image, and will display the entire thing at the coordinates you specify.

Creating the bitmap

In Allegro, images are BITMAPs. Without installing extra libraries, you can only use bitmap (.bmp) files in your game. Here is how you create a BITMAP object:

```
BITMAP *image;
```

Loading in the image

For most images, you will load in a file from outside...

```
image = load_bitmap( "gfx\\filename.bmp", NULL );
```

The NULL parameter is the palette, but you can keep it as NULL unless you're using a specific one.

But sometimes (like for the buffer) you will want to create one...

```
buffer = create_bitmap( 640, 480 );
```

Drawing the image

The code for drawing the image is as follows:

```
masked_blit( image, buffer, spritesheet_x, spritesheet_y,
     location_x, location_y, width, height );
```

Drawing the buffer to the screen

```
blit( buffer, screen, 0, 0, 0, 0, screenWidth, screenHeight );
clear_bitmap( buffer );
```

Blit's parameters are:

blit(source_bmp, dest_bmp, xsrc, ysrc, xdest, ydest, width, height)
(taken from Allegro Quick Reference)

If you don't use clear_bitmap afterwards, you will get an “etch-a-sketch” effect because it will be drawing on the same image each time, never erasing the previous one.

Destroying the bitmap

Always remember to do this or you will have memory leaks and your computer will slow down over time!

```
destroy_bitmap( image );
```

SDL code

Creating the surface

Defining the image itself is similar to with Allegro, except it's called an SDL_Surface instead of a BITMAP:

```
SDL_Surface *image;
```

Loading in the image and setting the alpha color

In SDL, we will load in the image, but then have to tell SDL which color is to be transparent. We can do these this way:

```
SDL_Surface *image = SDL_LoadBMP( "gfx\\player.bmp" );

// Set pink color transparent
Uint32 colorKey = SDL_MapRGB( image->format, 0xFF, 0, 0xFF );
SDL_SetColorKey(image, SDL_SRCCOLORKEY, colorKey );
```

Displaying the image

In SDL, it's blit function takes drawing and cropping coordinates from two SDL_Rect objects. location.x and location.y will be where the image is actually drawn on the buffer (or destination surface), and the crop.x, crop.y, crop.w, and crop.h denote what off the sprite sheet you'll draw.

```
SDL_BlitSurface( imageToDraw, &crop, buffer, &location );
```

Drawing the buffer to the screen

Once you draw all of the images that will be on screen to the **buffer**, use SDL Flip to draw the buffer to the screen.

```
SDL_Flip( buffer );
```

Destroying the surface

Always remember to do this or you will have memory leaks and your computer will slow down over time!

```
SDL_FreeSurface( image );
```

With SDL, you do not need to free the buffer. Since we use SDL_SetVideoMode on the buffer surface, SDL_Quit will free it for us automatically.

As you can see (if you're reading both sections o_o), this is one of the areas where Allegro seems a lot easier.

Animating your sprites

You probably want to have at least your character animate in the game. To do this, you should keep a float that stores the current frame. Each time the player moves, increment this by some small amount, like 0.5. Maybe have a function like this:

```
void Character::IncrementFrame()    // a function in the Character class
{
      frame += 0.5;
      if ( frame > maxFrame )
            frame = 0;
}
```

When you draw the sprite, you will set it's spritesheet x coordinate to

```
(int)frame*width;
```

So if frame were 2.5, it would draw frame 2 on the character sheet, which is at coordinates 2*width.

Check the section on Enums for using the direction to set the spritesheet y coordinate.

Lesson 4: Playing sound

Most games have sound and music. Without it, many games would seem unpolished. You can find public domain sound effects and music by doing a Google search.

Usually, wave files are sound effects and midi files should be the background music since wave takes up a lot of room.

Allegro code for wave files

You can also find other libraries for use with Allegro to let you load in other formats of sound, but on it's own Allegro just loads wave and midi files.

Creating a sample

```
SAMPLE *doggy;
```

Sample types are used for wave files in Allegro.

Loading in the sample

```
doggy = load_sample( "woof.wav" );
```

Playing the sample

```
play_sample( doggy, 255, 128, 1000, false );
```

The parameters for play_sample are:
play_sample(sample, volume, pan, frequency, loop)
(taken from Allegro Quick Reference)

Volume ranges from 0 to 255.
Pan is how much sound is played on the left/right speakers. 128 means equally (or "center", 0 is left speaker, 255 is rights peaker).
Frequency is playback rate, with 1000 being normal speed. 500 will be half speed and 2000 will be double speed.
If the loop parameter is true, the sound will continue to loop until `stop_sample( doggy );` is called.

Destroying the sample

Always destroy your sounds afterwards!

```
destroy_sample( doggy );
```

Allegro code for midi files

Creating a midi

Midis are similar to samples:

```
MIDI *bgsong;
```

Loading in the midi

```
bgsong = load_midi( "waffles.mid" );
```

Playing the midi

```
play_midi( bgsong, true );
```

First parameter is the midi file, and second one is whether it loops or not.

Destroying the midi

```
destroy_midi( bgsong );
```

SDL code for sounds

For SDL, you have to install extra library components to use sound, such as SDL_mixer. Check out LazyFoo's **Setting up SDL Extension Libraries** tutorial:
http://lazyfoo.net/SDL_tutorials/lesson03/index.php
and get SDL_mixer files from:
http://www.libsdl.org/projects/SDL_mixer/

Installing the mixer library generally will consist of copying contents between the include and lib folders (from the downloaded files to your IDE's directory).

By installing the SDL_mixer library, not only does it make it easier to play sounds, but you can also load waves, ogg, mod, mp3, and midi files.

Additional initialization and end of program code

At the beginning of any .h or .cpp files using SDL_mixer, we will need to have:

```
#include "SDL_mixer.h"
```

or

```
#include "SDL/SDL_mixer.h"
```

Depending on how you have it set up.

Now with SDL_mixer, we'll add this into the initialization function:

```
Mix_OpenAudio( 22050, MIX_DEFAULT_FORMAT, 2, 4096 );
```

The first parameter is the sound frequency, second is the format, third is the amount of sound channels (2 for stereo, 1 for mono), and fourth is sample size.

And, you'll need to add in the SDL_mixer.dll into your game directory.
If you want to use ogg files, you need to add libvorbis-0.dll, libogg-0.dll, libvorbisfile-3.dll.

At the end of your program, you'll have to additional

```
Mix_CloseAudio();
```

Once the program is done.

Mix_Chunk vs. Mix_Music

So we have Mix_Chunk and Mix_Music available. What is the difference?
The main difference is that the SDL_Mixer library has a special channel exclusively for music, and more specialized functions for music files.

Creating files

Creating each type is very similar -

```
Mix_Music *song;
Mix_Chunk *soundEffect;
```

If you want, you might set these to NULL initially.

Loading files

```
song = MIX_LoadMUS( "bgSong.ogg" );
soundEffect = Mix_LoadWAV( "sound.wav" );
```

Playing sounds

Music Files

```
Mix_PlayMusic( song, -1 );
```

Arguments are which Mix_Music file to play, and how many times to loop (-1 being infinite).

Chunk Files

```
Mix_PlayChannel( -1, soundEffect, 0 );
```

Arguments are channel to play on (-1 for first available one), the Mix_Chunk file, and how many times to loop (-1 being infinite).

Pausing and resuming files

Music Files

```
Mix_PauseMusic();
Mix_ResumeMusic();
```

Chunk Files

```
Mix_Pause( -1 );
Mix_Resume( -1 );
```

The argument is which channel to stop. Use -1 to stop all Chunk channels.

Stopping sounds

Music Files

```
Mix_HaltMusic();
```

Chunk Files

```
Mix_HaltChannel( int channel );
```

The argument is which channel to stop. Use -1 to stop all Chunk channels.

Freeing files

Once you're done with the program, free all your Mix_Chunk and Mix_Music files!

```
Mix_FreeMusic( song );
Mix_FreeChunk( soundEffect );
```

Lesson 5: Displaying text

In Allegro, drawing text to the screen is very easy. All you have to do is call one function and it draws it out for you. However, in SDL, you have to render text to a surface on your own.

Allegro textprintf functions

From the Allegro Quick Reference...

```
textprintf_ex(bmp, font, x, y, color, -1, string, …);
textprintf_centre_ex(bmp, font, x, y, color, -1, string, ...);
textprintf_right_ex(bmp, font, x, y, color, -1, string, ...);
```

Each of these aligns a little different. Centre will draw the center of your string of text at your x, y coordinates. In your string, you can have a %f, %i, or %s placeholder, and then include a variable name as another parameter, like this:

```
textprintf_ex(buffer, font, 0, 0, makecol( 255, 255, 255 ), -1, "%s
     score: %i, %f mph", player.name, player.score,
     (float)(player.score/totalGameTime) );
```

If you want to use the default font, keep “font” as the font parameter.
This outputs an 8x8px font.

For the color parameter, you'll use the function:

```
makecol( r, g, b );
```

Where r, g, and b are from 0 to 255.

And the -1 parameter is the background color. Keep this as -1 if you want it to be transparent.

Drawing text in SDL

Before we display text in SDL, we have to install another extra library in SDL for fonts.

You can get the extended library from here:
http://www.libsdl.org/projects/SDL_ttf/

and you can find LazyFoo's tutorials on installing extension libraries here:
http://lazyfoo.net/SDL_tutorials/lesson03/index.php

Additional initialization and end of program code

At the beginning of any .h or .cpp files using SDL_ttf, we will need to have:

```
#include "SDL_ttf.h"
```

or

```
#include "SDL/SDL_ttf.h"
```

Depending on how you have it set up.

In your initialization function, you will need to add:

```
TTF_Init();
```

And when your program is done running, you need to add:

```
TTF_Quit();
```

Writing a function to display text

Since SDL handles text by rendering it to a SDL_Surface, it'd be best to write a function to draw text to a particular coordinate so you can call that quickly instead of having to create a coordinate SDL_Rect, SDL_Color, etc. The following is my function to write text, and you are free to use or modify as you wish:

```
void DrawText( SDL_Surface *destination, string msg, int x, int y,
      int size, SDL_Color color )
{
    TTF_Font *font;
    font = TTF_OpenFont( "data/georgia.ttf", size );
    if ( !font )
        cerr<<"Error loading georgia.ttf"<<endl;

    SDL_Rect coordinates;
    coordinates.x = (int)x;
    coordinates.y = (int)y;

    SDL_Surface *message = NULL;
    message = TTF_RenderText_Solid( font, msg.c_str(), color );

    SDL_BlitSurface( message, NULL, destination, &coordinates );

    TTF_CloseFont( font );
    SDL_FreeSurface( message );
}
```

Lesson 6: Regulating FPS

Now just a minute! We haven't learned everything we need to know to make a game yet, even though we now know about graphics, sound, input, and drawing text! We also need to regulate the FPS. Why?

Have you ever popped in an old PC game, only to find it runs way too fast? (ie Sonic 3 & Knuckles for PC is the only game I can think of that does that to me). We need to regulate the FPS using a timer so that it won't run at different speeds on different grades of computers.

Unless you just want to spite people with better PCs than you and make it harder for them to play because the game is too fast... :'(

Allegro timer

Code from Loomsoft, altered by me

```
volatile long fps = 0;
void IncFps() { fps++; }

int main()
{
	// Do the Allegro initialization BEFORE the timer locks
	LOCK_VARIABLE( fps );
	LOCK_FUNCTION( IncFps );
	install_int_ex( IncFps, BPS_TO_TIMER(90) );
	// The rest of the program goes here
```

SDL timer

We will want to create an integer that stores the “Tick” the program is at, and use that to compare the program's speed to the maximum FPS. If necessary, we'll use the SDL_Delay function to throttle the program down to a normal speed.

```
int currentTick;

while ( !done )
{
	currentTick = SDL_GetTicks();
	// ... Program stuff

	// Check FPS rate
	if ( SDL_GetTicks() - initialTick < 1000 / maxFPS )
	{
		SDL_Delay( (int)((1000/FPS)-GetTicks()) );
	}
}
```

Lesson 7: Enums and random numbers

Just a quick look at them in case you didn't cover them in C++.

Enums

Enumerations are a handy way of representing numbers. Numbers are a lot smaller in size than strings. Maybe you want to use a number to represent the direction a player is pointing? Using a string would take too much space, and while you COULD use a char like 'r', 'u', 'd', 'l', it's much easier to read if it's something like if (direction == UP)

Enums look like this:

```
enum Direction { DOWN, UP, LEFT, RIGHT };
```

or

```
enum Direction { DOWN = 0, UP = 1, LEFT = 2, RIGHT = 3 };
```

If you don't assign a number value to each one, it will still be assigned an arbitrary number. The reason you may use numbers for directions are to tell where in the tilestrip the image is.

This way, when you draw the character, you will crop the sprite out like this:
spritesheet_x = frame * width;
spritesheet_y = currentDir * height;

Random numbers

Random numbers. What are they used for? Well, a lot. In this game, our stick will get random coordinates every time it's picked up. In other games, it may generate an enemy location or even be used for AI (What do I feel like doing today? *rand()* I'm going to do option #3).

Seeding – what is it, and how do we do it

Since random numbers aren't truly random and have to have a "seed" number to generate the numbers from, seeding is usually done to the current time. If you seed to a constant number, you'll find that each time you run the program, you'll get the same random numbers in the same order. You may want this, and you may not.

So here's how you seed to the time:

```
srand( (unsigned)time(NULL) );
```

Generating random numbers

Now to actually generate the random numbers, you will use rand(). However, this will create a number between 0 and 1 (some long decimal number), so how do you get it between the numbers you want?

```
x = (int)rand() % 10;
```

This code will generate a random number between 0 and 9. If you want it from 1 to 10, then add +1 to the end of the expression.

% is for modulus, and is pretty interesting. It's basically an expression to get the remainder from `(int)rand()` divided by `10`. It also makes sure that if the value is above 10, it "wraps around" back to 0 and continues.

If you want to generate random coordinates for the stick, it should look something like this:

```
x = ( (int)rand() % 608 );
y = ( (int)rand() % 448 );
```

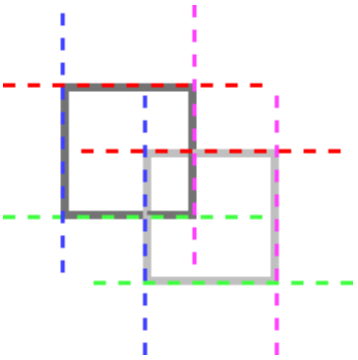
The screen is 640x480. If the stick's x coordinate gets generated as 640, then it's left side will be at that value and it will essentially be off the screen. This is why we subtract 32 from each of the values.

Lesson 8: Bounding-box collision detection

Bounding-box collision detection is just one of many ways to detect collision between objects. It consists of having each collidable object have a region that is “solid”. This can either be the entire picture, but is probably better if it's a smaller region of the image.

A **B** **C**

The Ayne sprite is 32x32, but the region should probably be from (7, 1) to (37, 47) (image B) or (11, 8) to (34, 47) (image C) if you don't want things like her hair and the ends of her arms causing collisions.



Left edges = blue, Right edges = pink,

Top edges = red, Bottom edges = green

The basic idea for this collision is as such:

Left1 < Right 2 && Right1 > Left2 &&

Top1 < Bottom2 && Bottom1 > Top2

You should probably write a function somewhere in main.cpp or it's own header file to return a bool for IsCollision, rather than having the function be part of a class. The parameters should be two Rectangles.

Allegro code:

First you need to write a Rectangle struct to hold the collisions, like this:

```
struct Rectangle
{
	int x, y, w, h;
}
```

Shared code:

Before you write the collision code, you'll need to make sure your characters and objects in the game have a collision region Rectangle. This way, you only pass the rectangle to the `IsCollision` function so it can be used with all different types of objects.

Also, you'll have to write some functions, because if you just pass the collision region coordinates, it will be based at 0,0 and you won't get the right results. You need to add the collision region to the object's current coordinates.

In Allegro, you will have to write your own Rectangle struct, but in SDL you can use SDL_Rect. For the following code, it's assuming you're using Allegro, but just replace Rectangle with SDL_Rect if you're using SDL.

```
#include "Rectangle.h"

class Character		// Incomplete Character class
{
	private:
		Rectangle colRegion;
		Rectangle coordinates;
	public:
		Rectangle RegionCoordinates();
};

Rectangle Character::RegionCoordinates()
{
	// create a rectangle to pass to the IsCollision function.
	Rectangle temp;
	temp.x = colRegion.x + coordinates.x;
	temp.w = colRegion.w;
	temp.y = colRegion.y + coordinates.y;
	temp.h = colRegion.h;

	return temp;
}
```

Here's how you would implement and call the `IsCollision` function (again, just replace Rectangle with SDL_Rect for SDL):

```
// ...includes go here...

bool IsCollision( Rectangle A, Rectangle B );

int main()
{
   // ...stuff...
   if (IsCollision(player.RegionCoordinates(), stick.RegionCoordinates()))
   {
      player.AddToScore( 1 );
   }
   // ...more stuff...
}

bool IsCollision( Rectangle A, Rectangle B )
{
      if (  A.x         <     B.x + B.w   &&
            A.x + A.w   >     B.x         &&
            A.y         <     B.y + B.h   &&
            A.y + A.h   >     B.y )
      {
            return true;
      }
      return false;
}
```

Lesson 9: Planning the game

Now you know the basic building blocks of writing the game.
Even though this game is going to be very basic, it's good to get into the habit of planning various elements of the game, from general ideas you want to keep in mind, what graphics you'll need, and how you want to structure it. I like to write design documents with google docs because I can access them anywhere, too. Google docs are also share-able, so that multiple people can work on a design doc together.

Also keep in mind that your game doesn't have to be *Pickin' Sticks*. This game just has a character collecting objects. Your game can have a cat collecting fish, or a peanut butter sandwich collecting shoe stores. It's up to you!

Sample Pickin' Sticks design document

PICKIN' STICKS

(Or whatever you want to call it)
Design Document

[Table of Contents]

General ideas for features and goals
(sort of a brain-storming section that you flesh out later on)

- A basic graphical game with collision detection
- After you pick up certain amounts of sticks, you “rank up”

Story Elements

Game setting

- The game is set in a grassy field. Maybe after the grass is implemented, a fence perimeter will be added, and after that the back of a house, so that you feel like you're in someone's back yard picking up sticks.

Game characters

- [Your name for character]

Resources

Graphics

- Grassy background – either tiled grass or one big background image
- Alec
  - Left, Right, Up, Down directions, each with 3 frames of animation
- A stick
- The title screen

Sound effects

- Noise when stick is picked up (jingly?)
- Walking noise when character is moved?
- “Thud” noise when player tries to walk off the screen?

Music
Music will be out of place. Perhaps up-beat epic battle music, or a slow sad ballad.

- Title screen song
- Main gameplay song

Coding Specifics

Game Objects

- Rectangle class (if using Allegro, otherwise just use SDL_Rect)
- Player class
  - Holds x, y, coordinates and width and height
  - Holds direction player is facing
  - Holds frame of animation it's currently on
  - Has a Rect object for the collision region
  - Has a Points integer to store player's score
  - Has IncrementScore() function, and appropriate get/set functions
  - a Rank function that will return a string of what the player's rank is based on points / amount of sticks picked up
- Stick class
  - Holds x, y coordinates and width and height
  - Has function to randomly generate coordinates
  - Has a Rect object for the collision region
- System class
  - handles Allegro/SDL initialization, fullscreen toggle, game state
- ImageManager class **(optional)**
  - Holds all of the images in the game. Instead of keeping them in main and passing them to different functions, we will pass the instance of imageManager instead
- SoundManager class **(optional)**
  - Similar to the ImageManager class

Al Lowe, creator of Leisure Suit Larry, Torin's Passage and Freddy Pharkas also has his old design docs for those games on his website (www.allowe.com). You may check them out for ideas and inspiration.

Lesson 10: Super-basic game structure

In this guide, a lot of the example code is bare-bones. If you write your program with the example code as-is, it will not be very object oriented or neat. I will outline some example classes for things like Input and Timers here.

The game loop

Your game's logic will run something like this:

1. Initialize library and game
2. While player hasn't quit
    1. Check for and handle input
    2. Check for and handle collisions
    3. Draw everything to buffer
    4. Draw buffer to screen
    5. Clear buffer
    6. Loop
3. Once player has quit, clean up resources
4. return 0

The main objects

These should definitely be classes, and not just loose variables sitting in main. If you notice any similarities, you might use some inheritance.

Player

- A rectangle holding the x and y coordinates, and player width and height
- An enumeration of the current direction player is facing
- Current frame #
- Walk speed
- Score
- Move function
- Draw function
- Increment frame function
- Increment score function
- Necessary Get and Set functions

Stick

- A rectangle holding the x and y coordinates, and stick width and height

- Generate coordinates function
- Necessary Get and Set functions

The helper objects

These keep your code cleaner if you put them in classes, but they're not necessary for such a basic game. You *can* put all your image and sound files in main.cpp, but it's not something you want to get into the practice of.

ImageManager

- All the different images in the game
  - Player
  - Grass
  - Stick
  - Buffer
- Setup function or constructor to load in all the images when the game begins
- Destructor to destroy all files when program ends
- Function to draw buffer to screen and then clear it

SoundManager

- All the different sounds in the game
  - Stick picked up happy victory noise!
  - Background song
- Setup function or constructor to load in all the sounds when game begins
- Desctructor to destroy all files when program ends
- Function to play a sound
- Function to stop a/all sounds.

System

- The screen width and height
- Function that calls Allegro initialization functions
  - allegro_init();
  - install_keyboard();
  - install_timer();
  - install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, 0);
  - set_color_depth(16);
  - set_gfx_mode(GFX_AUTODETECT, scrWidth, scrHeight, 0, 0);

Wrapping up

Your turn!

I've outlined the basics of what you'll need to know to make a Pickin' Sticks game, so now it's your turn to try it! Try to get as far as you can, and if you're stuck, DEFINITELY ask questions! I will post video responses and update with answers. Every question asked helps out everybody.

My code in this tutorial is public domain, you can do whatever with the games you write based on it, with no credit to me needed. Same with the public domain graphics and sound.

Distributing your game

When your game is done, you should organize your folder a little bit so it's nice and clean. Copy-paste a copy of your folder, this one will be for distribution. If you want to have your source code available, create a “src” or “source” folder and move all the .h and .cpp files into it. Do not include your project file, just the .h and .cpps!

Make sure you have alleg42.dll or SDL.dll in the folder that has the main .exe file.

Remember to write a readme file! It should have the game's purpose, controls, and your credits!

And...

Let me know about your game! If you want, I can host it on the website under this tutorial's page!

T-shirts, anyone?

If this tutorial has been helpful to you, or if you have enjoyed other things available on my website, please consider donating or buying a product from the Moosader store at

www.moosader.com

Every little bit helps! Thanks for your support!