



# SDL (Simple DirectMedia Layer) with C

Knoxville Game Design

December 2019

Levi D. Smith

# SDL Overview

- Created by Sam Lantinga (Blizzard, Valve, World of Warcraft)
- Bindings for several languages (C, Ruby, Python, Lua, C#, etc)
- First release 1998, Latest release July 2019

# Games using SDL



Also many Linux ports!



KNOXVILLE  
GAME  
DESIGN

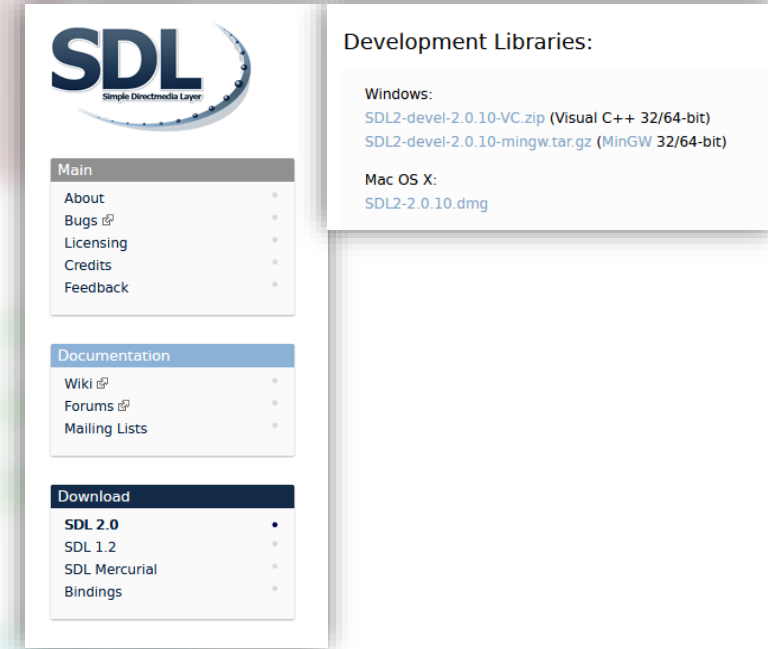
# Pros and Cons (SDL with C)

- Pros
  - Low level control
  - Only use what you need
  - Fast (compile times)
  - Small executables for small games
  - Free (zlib license)
  - Multi-platform (Windows, Mac, Linux, ...)
  - 2D and 3D
- Cons
  - Complex setup and API (compared to Unity, GameMaker, etc)
  - Possible memory leaks / No garbage collection
  - Most game frameworks/engines hide the messy details
  - Anything aside the basic operations requires additional libraries



# Getting Started

- Download development libraries from [libsdl.org](http://libsdl.org)
  - These examples will use the MingGW libraries for Windows
  - [SDL2-2.0.10-mingw.tar.gz](http://www.libsdl.org/release/SDL2-2.0.10-mingw.tar.gz)
- Download and install MinGW
  - <http://mingw.org/>
- In MinGW Installation Manager, ensure that mingw32-gcc-bin is selected and installed
- Run `MinGW\msys\1.0\msys.bat`



The screenshot shows the MinGW Installation Manager window. The 'All Packages' tab is selected, and the 'mingw32-gcc-bin' package is checked and highlighted in green. The table below lists the available packages.

Package	Installed Version	Repository Version	Description
<input type="checkbox"/> mingw32-binutils-info		2.32-1	The GNU Binary File Utilities
<input type="checkbox"/> mingw32-binutils-lang		2.32-1	The GNU Binary File Utilities
<input type="checkbox"/> mingw32-binutils-man		2.32-1	The GNU Binary File Utilities
<input checked="" type="checkbox"/> mingw32-gcc-bin	8.2.0-5	8.2.0-5	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-dev		4.8.2-2	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-doc		4.8.1-5	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-info		8.2.0-5	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-lang		8.2.0-5	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-lic		8.2.0-5	The GNU C Compiler
<input type="checkbox"/> mingw32-gcc-man		8.2.0-5	The GNU C Compiler

# Test Your C Compiler

- Create a simple “Hello World” program
- Compile it with GCC
- Should create an executable in the working directory
- Some virus software may flag the executable, so restore it if needed

```
MINGW32:/d/ldsmith/presentations/sdl/test
#include <stdio.h>

int main(void) {

    printf("Hello World\n");

    return 0;
}
~
~
~
```

```
MINGW32:/d/ldsmith/presentations/sdl/test

gatec@darknut /d/ldsmith/presentations/sdl/test
$ vi test.c

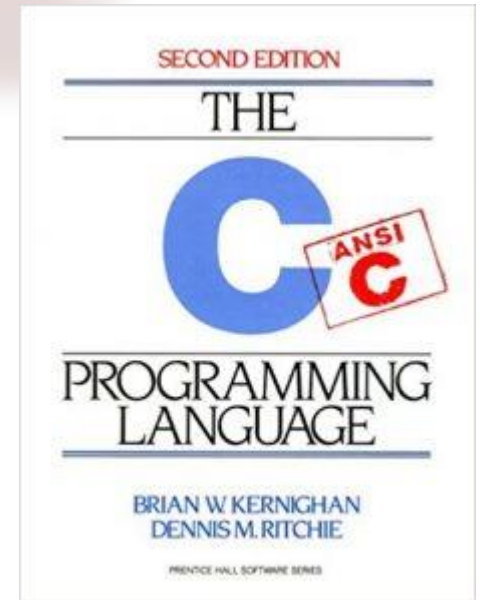
gatec@darknut /d/ldsmith/presentations/sdl/test
$ gcc -o test test.c

gatec@darknut /d/ldsmith/presentations/sdl/test
$ ./test
Hello World
```

**KNOXVILLE**  
**GAME**  
**DESIGN**

# C differences from other Languages

- No *const*, #define for constants
- No bool, use int and define TRUE/FALSE constants to 1/0
  - if / while statements – expression evaluation to 1 is true; evaluation to 0 is false
  - Note - Returning 0 is success, non-zero is failure
- Pointers / Structures instead of Objects / References
- Define function prototypes to avoid compile time errors
- Structs can hold multiple data types like an object
- No List / Vector, but you can make your own linked lists with pointers
- Don't assume that variables are initialized to 0 / FALSE
- *char \** instead of *string*
- No inheritance / subclasses



KNOXVILLE  
GAME  
DESIGN

# C Pointers

- *int \*p* – Create a pointer to an int
- *p* – The address of *p* in memory
- *\*p* – The value that *p* is pointing at
- Use malloc / free in stdlib.h to allocate and free memory
- dereferencing a NULL pointer is bad (core dumps)
- Assigning value larger than allocated memory is bad (segmentation faults)
- Use *sizeof(<type>)* to get the amount of memory required for data type
- Use *&e* to get the address of a non-pointer variable
- SDL has it's own memory allocation methods, but you still need to know how to work with pointers

```
MINGW32:/d/ldsmith/presentations/sdl/test
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("Pointer demo\n");

    printf("Create an int pointer\n");
    int *p1;

    printf("Allocate memory for an int\n");
    p1 = malloc(sizeof(int));

    printf("Assign the value that p1 is pointing at to 42\n");
    *p1 = 42;

    printf("Value of *p1 is: %d\n", *p1);

    printf("Address of p1 in memory is: %d\n", p1);

    printf("Free the memory allocated to p to prevent memory leak\n");
    free(p1);

    printf("Define an integer (non pointer) p2\n");
    int p2 = 1885;

    printf("Point p1 to the address of p2 (&p2)\n");
    p1 = &p2;

    printf("Print the value that p1 is pointing at (*p1): %d\n", *p1);

    return 0;
}
```



# Setting up SDL libraries

- Start MinGW
- Copy the library tar file to a temporary directory
- Extract the tar file
- change to SDL2-2.0.10 directory
- Run “make native”

```
$ mkdir lib  
$ cd lib/  
$ mv /d/Downloads/SDL2-devel-2.0.10-mingw.tar.gz .  
$ tar xvf SDL2-devel-2.0.10-mingw.tar.gz  
$ cd SDL2-2.0.10/  
$ make native
```

# Creating a window

- Initialize SDL with *SDL\_Init*
- Create a window with *SDL\_CreateWindow*
- Access the screen surface with *SDL\_GetWindowSurface*
- Draw a filled rectangle with *SDL\_FillRect*
- Update with *SDL\_UpdateWindowSurface*
- Cleanup with *SDL\_DestroyWindow*
- Exit with *SDL\_Quit*

```
MINGW32:/d/ldsmith/presentations/sdl/test
#include <SDL.h>
#include <stdio.h>

#define SCREEN_WIDTH 640
#define SCREEN_HEIGHT 480

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

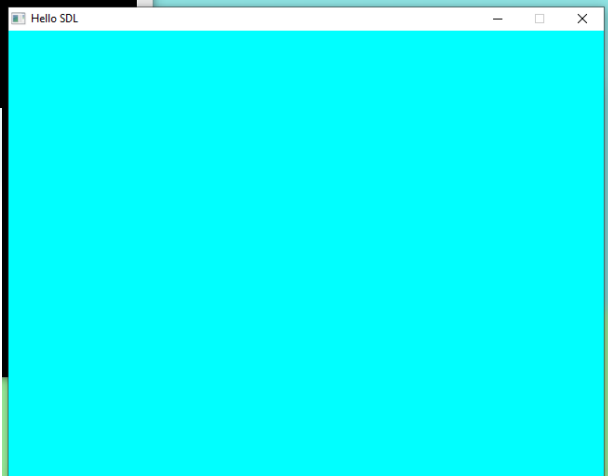
    SDL_Window* window = NULL;
    SDL_Surface* screenSurface = NULL;
    if (SDL_Init( SDL_INIT_VIDEO) < 0) {
        printf("Error: %s\n", SDL_GetError());
        return 1;
    }

    window = SDL_CreateWindow("Hello SDL", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, SCREEN_WIDTH, SCREEN_HEIGHT,
    SDL_WINDOW_SHOWN);
    if (window == NULL) {
        printf("Error: %s\n", SDL_GetError());
        return 1;
    }

    screenSurface = SDL_GetWindowSurface(window);
    SDL_FillRect(screenSurface, NULL, SDL_MapRGB( screenSurface->format, 0x00, 0xFF, 0xFF));
    SDL_UpdateWindowSurface( window);
    SDL_Delay(2000);

    SDL_DestroyWindow(window);
    SDL_Quit();

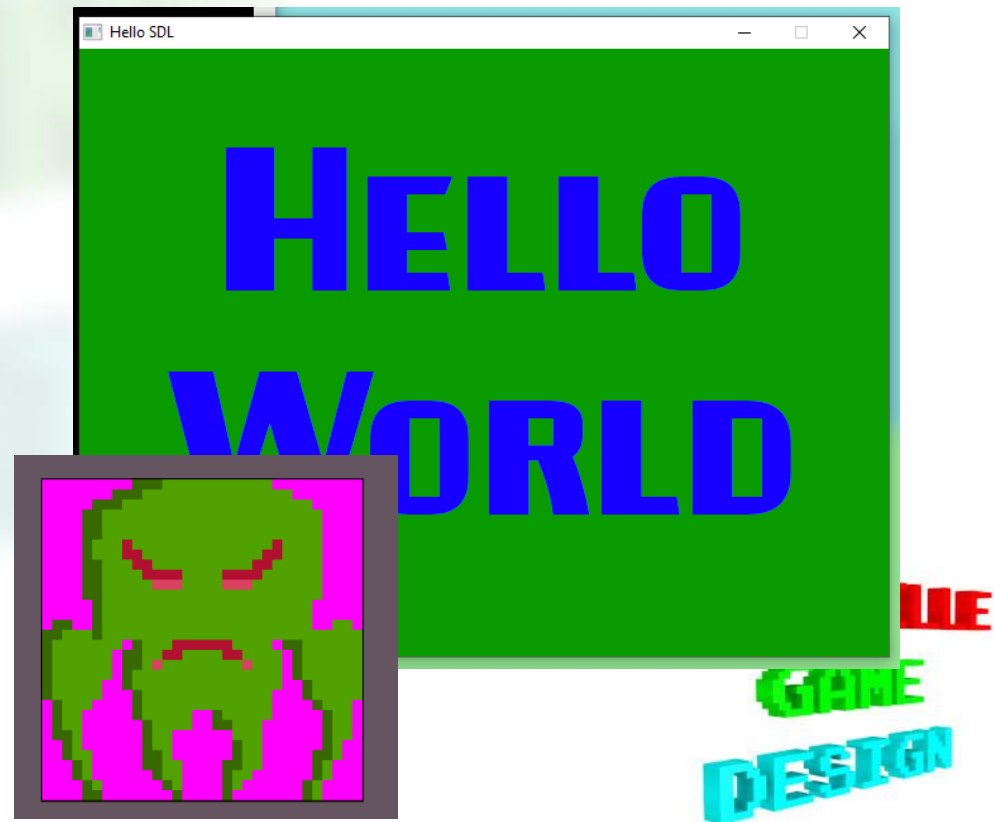
    return 0;
}
```



# Displaying Surfaces (“Sprites”)

- Use *SDL\_LoadBMP* to load a bitmap sprite
  - Can only load BMP files by default
  - Can load PNG with SDL\_image library (additional library that must be installed)
  - Can set color key (transparency) with *SDL\_SetColorKey*
  - Recommend magenta (255, 0, 255) for transparency color
- Draw to the screen surface with *SDL\_BlitSurface*
- Note – Use *SDL\_Renderer/SDL\_Texture/SDL\_RenderCopy* for hardware acceleration (topic for another time)

```
SDL_Surface* sprHello;  
sprHello = SDL_LoadBMP("hello.bmp");  
SDL_BlitSurface(sprHello, NULL, screenSurface, NULL);  
  
SDL_UpdateWindowSurface( window);  
SDL_Delay(10000);  
  
SDL_FreeSurface(sprHello);
```

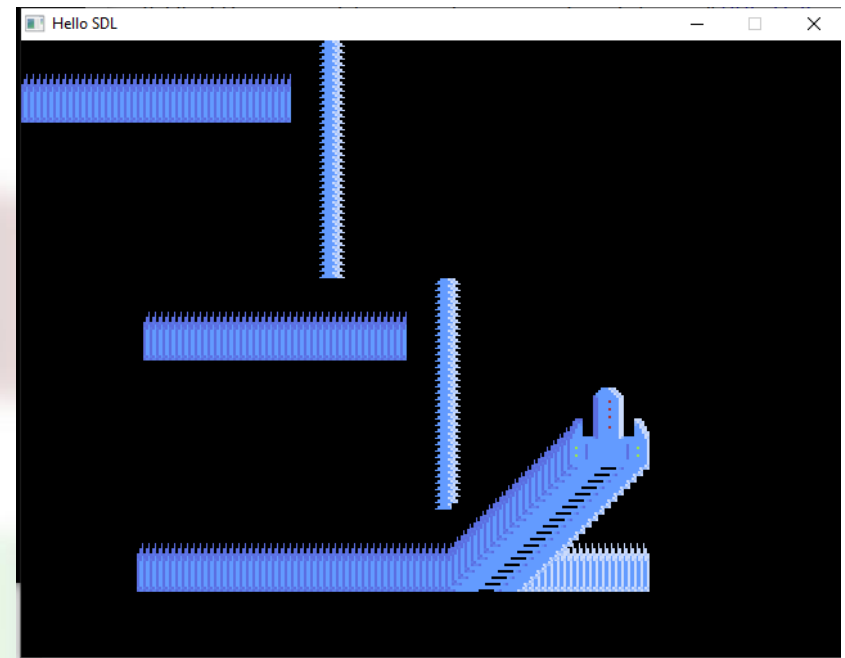


# Game Loop

- Create a loop that calls *SDL\_PollEvent(&e)*, where *e* is an *SDL\_Event*
- Stop looping when *e.type* equals *SDL\_QUIT*
- Suggest making an *update/draw* methods to handle input / drawing
- Check *e.type == SDL\_KEYDOWN* to detect key press
- Use *e.key.keysym.sym* to get the pressed key
- Add *SDL\_Delay* to keep from tying up CPU
  - Calculate delay time with *SDL\_GetTicks* (minus time since last loop)

# Moving a Ship

- Create an *SDL\_Rect* to hold x, y position
  - Will be used as the destination parameter of *SDL\_BlitSurface*
  - Setting all Rect members (x, y, w, h) is important
  - Use *SDL\_UpdateWindowSurface* to update the screen after finished drawing (double buffering)
- Must clear the screen yourself, otherwise “smearing” will occur
  - *SDL\_RenderClear*, unless redrawing the entire screen
- Struct for holding ship values
  - x, y, vel\_x, vel\_y, isAlive



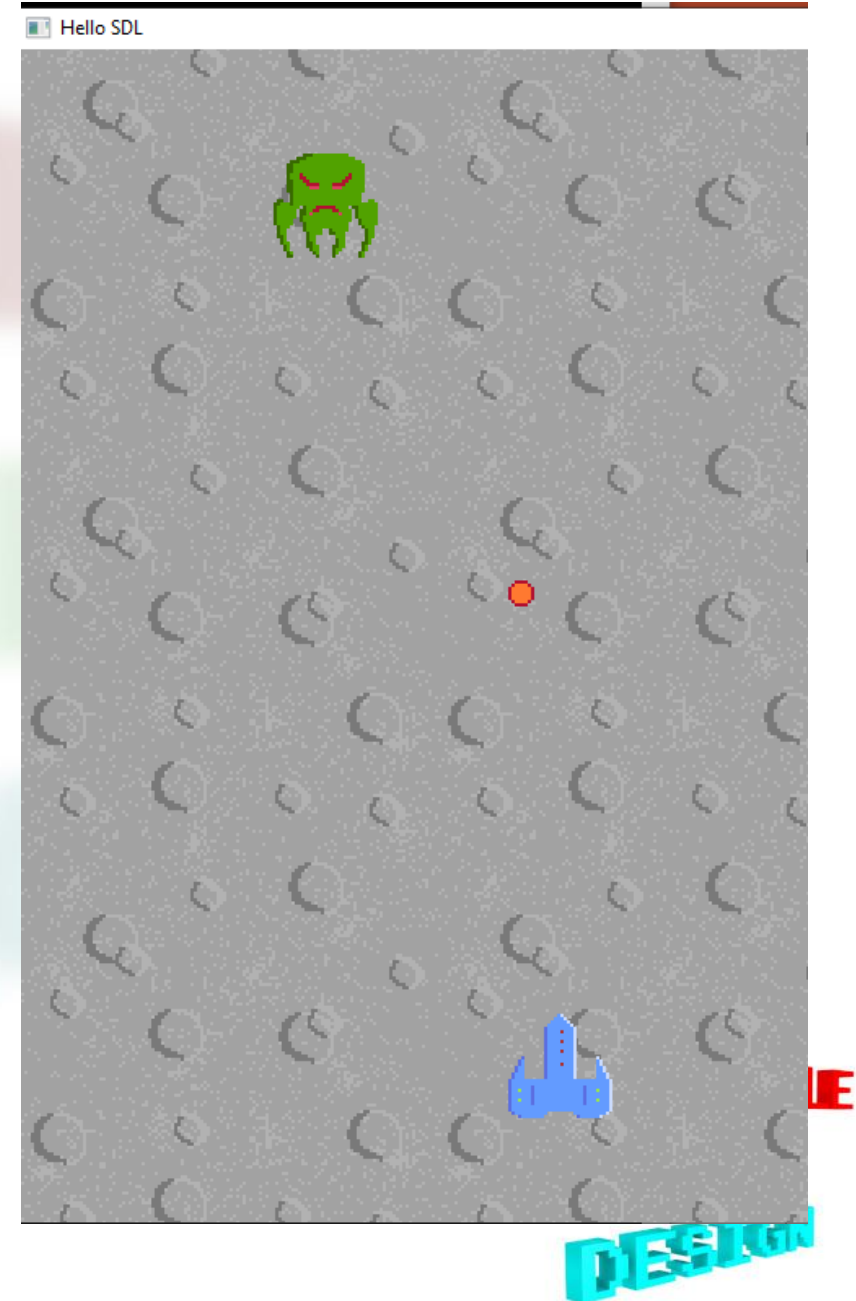
```
struct Ship {  
    float vel_x;  
    float vel_y;  
    int x;  
    int y;  
    int width;  
    int height;  
} ship;
```

```
void handleInput(int iType, int iKey) {  
    float fSpeed = 1;  
  
    if (iType == SDL_KEYDOWN) {  
        if (iKey == SDLK_UP) {  
            ship.vel_y = -fSpeed;  
        } else if (iKey == SDLK_DOWN) {  
            ship.vel_y = fSpeed;  
        } else if (iKey == SDLK_LEFT) {  
            ship.vel_x = -fSpeed;  
        } else if (iKey == SDLK_RIGHT) {  
            ship.vel_x = fSpeed;  
        } else if (iKey == SDLK_q || iKey == SDLK_ESCAPE) {  
            iKeepLooping = FALSE;  
        }  
    }  
  
    if (iType == SDL_KEYUP) {  
        if (iKey == SDLK_UP && ship.vel_y < 0) {  
            ship.vel_y = 0;  
        } else if (iKey == SDLK_DOWN && ship.vel_y > 0) {  
            ship.vel_y = 0;  
        } else if (iKey == SDLK_LEFT && ship.vel_x < 0) {  
            ship.vel_x = 0;  
        } else if (iKey == SDLK_RIGHT && ship.vel_x > 0) {  
            ship.vel_x = 0;  
        }  
    }  
}
```



# Adding Enemies and Bullets

- Add an enemy structure
  - x, y, health, lifetime, isAlive
- Back and forth movement
- Add a bullet structure
  - x, y, isAlive
  - Set isAlive to false if its y position is less than zero
  - Only draw/update if isAlive is TRUE
- Get things working with one enemy and one bullet before adding multiples



# Collision Detection

- Check collision between bullet and enemy
- Check collision between enemy and ship
- Set isAlive property to FALSE on collision
- Types of collision
  - Box/Rectangle
  - Circle (Pythagorean,  $a^2+b^2=c^2$ )
  - Pixel Perfect
  - Continuous (fast moving objects)

Very simple box collision

```
void checkCollisions() {  
    if ( (enemy.isAlive) &&  
        (bullet.x >= enemy.x && bullet.x < enemy.x + enemy.width) &&  
        (bullet.y >= enemy.y && bullet.y < enemy.y + enemy.height) ) {  
        bullet.isAlive = FALSE;  
        enemy.isAlive = FALSE;  
    }  
  
    if ( (enemy.isAlive) && (ship.isAlive) &&  
        (ship.x >= enemy.x && ship.x < enemy.x + enemy.width) &&  
        (ship.y >= enemy.y && ship.y < enemy.y + enemy.height) ) {  
        ship.isAlive = FALSE;  
    }  
}
```

# Multiple enemies / bullets

- Can use array, but not optimal
  - Array is basically a set of pointers
- Change variable from *struct Enemy enemy* to *struct Enemy \*enemy*
- Can dereference the pointer and access structure member with ->
  - `struct Enemy *enemy;`
  - `enemy = malloc(sizeof(struct Enemy));`
  - `printf("x: %d, y: %d\n", enemy->x, enemy->y);`
- `enemy->x` is the same as `(*enemy).x`

```
struct Enemy {
    int iHealth;
    int x;
    int y;
    int width;
    int height;
    float vel_x;
    float vel_y;
    float flifetime;
    int isAlive;
};
struct Enemy *enemy;
struct Enemy enemyList[5];
```

```
struct Enemy e1;
e1.x = 640;
e1.y = 128;
e1.width = 64;
e1.height = 64;
e1.isAlive = TRUE;

struct Enemy e2;
e2.x = 640;
e2.y = 128 + (64 * 1);
e2.width = 64;
e2.height = 64;
e2.isAlive = TRUE;

struct Enemy e3;
e3.x = 640;
e3.y = 128 + (64 * 2);
e3.width = 64;
e3.height = 64;
e3.isAlive = TRUE;

struct Enemy e4;
e4.x = 640;
e4.y = 128 + (64 * 3);
e4.width = 64;
e4.height = 64;
e4.isAlive = TRUE;

struct Enemy e5;
e5.x = 640;
e5.y = 128 + (64 * 4);
e5.width = 64;
e5.height = 64;
e5.isAlive = TRUE;

enemyList[0] = e1;
enemyList[1] = e2;
enemyList[2] = e3;
enemyList[3] = e4;
enemyList[4] = e5;
```

```
struct Enemy e1;
addEnemy(&e1, 0, 128);

struct Enemy e2;
addEnemy(&e2, 0, 128 + (64 * 1));

struct Enemy e3;
addEnemy(&e3, 0, 128 + (64 * 2));

struct Enemy e4;
addEnemy(&e4, 0, 128 + (64 * 3));

struct Enemy e5;
addEnemy(&e5, 0, 128 + (64 * 5));

enemyList[0] = e1;
enemyList[1] = e2;
enemyList[2] = e3;
enemyList[3] = e4;
enemyList[4] = e5;
```

```
void addEnemy(struct Enemy *e, int init_x, int init_y) {
    e->x = init_x;
    e->y = init_y;
    e->vel_x = 0.2;
    e->vel_y = 0;
    e->width = 64;
    e->height = 64;
    e->fChangeMovementCountdown = 60 * 10;
    e->isAlive = TRUE;
}
```

GAME  
DESIGN

# Linked Lists

- Similar to Vectors/List collections
- Reference to the head of the list
- Each element has a pointer to the next element
- Iterate through the remaining elements of the list
  - Stop when the next element is NULL
- Add elements by iterating to the end of the list and setting the next element
- Remove an element by pointing the previous element to the next, and then free'ing the element to be removed
- Use void \* pointer for anonymous data types (must cast when accessing)

# Drawing Text

- Download the Windows MinGW development library
  - [https://www.libsdl.org/projects/SDL\\_ttf/](https://www.libsdl.org/projects/SDL_ttf/)
  - SDL2\_ttf-devel-2.0.15-mingw.tar.gz
- Run *make native* from the extracted directory
- Compile with
  - `gcc -o test_game test_game.c `sdl2-config --cflags --libs` -lSDL2_ttf`
- Test by adding `#include <SDL_ttf.h>` and `TTF_Init()` to your game
- SDL\_ttf docs - [https://www.libsdl.org/projects/SDL\\_ttf/docs/index.html](https://www.libsdl.org/projects/SDL_ttf/docs/index.html)
- Be sure to free your text surfaces to prevent memory leak
  - Check memory usage in Task Manager
  - More efficient to use one surface/texture with all characters, instead of allocating a new image on each draw
- Build a string with `sprintf` (from `string.h` library)



```
#include <SDL_ttf.h>
```

```
TTF_Font *fontDefault;
```

```
if (TTF_Init() == -1) {  
    exit(1);  
}  
fontDefault = TTF_OpenFont("SudburyBasin-Regular.ttf", 20);
```

```
SDL_Color colorText;  
colorText.r = 255;  
colorText.g = 255;  
colorText.b = 0;  
SDL_Surface *sprText = TTF_RenderText_Solid(fontDefault, "Hello World", colorText);  
  
pos.x = 100;  
pos.y = 100;  
SDL_BlitSurface(sprText, NULL, screenSurface, &pos);
```

```
SDL_FreeSurface(sprText);
```

▼	test_game.exe (32 bit)	2.9%	254.2 MB
	SDL Shooter		

```
char strScore[64];  
sprintf(strScore, "Score: %d", iScore);  
SDL_Surface *sprText = TTF_RenderText_Solid(fontDefault, strScore, colorText);
```

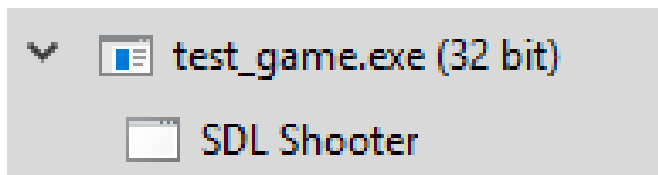


# Sound and Music

- Use SDL\_mixer to load and play audio
  - Download the MinGW development libraries from [https://www.libsdl.org/projects/SDL\\_mixer/](https://www.libsdl.org/projects/SDL_mixer/)
  - Extract SDL2\_mixer-devel-2.0.4-mingw.tar.gz
  - make native
  - Add `-lSDL2_mixer` to compile flags
  - Add `#include <SDL_mixer.h>` to source code
- Mix\_OpenAudio to initialize audio
- Mix\_LoadWAV to load files (.wav files only)
- Mix\_PlayMusic to play music
- Mix\_PlayChannel to play sound

# Distributing Your SDL Game

- Package your exe, image files (bmp), and SDL2.dll
  - Use the SDL2.dll SDL2-2.0.10-win32-x86.zip
  - Include any other DLLs. Use runtime libraries (such as SDL2\_ttf.dll, zlib1.dll, libfreetype-6.dll from SDL2\_ttf-2.0.15-win32-x86.zip)
  - Include any TTF font files and WAV audio files
- How to use 64 bit libraries? Requires MinGW-w64 or Visual Studio?
- Mixing 64 bit libraries with 32 bit executable probably won't work



KNOXVILLE  
GAME  
DESIGN

# Managing Code

- Can put code in multiple files (ship.c, enemy.c, bullet.c, etc)
  - Can simulate object oriented programming
  - Put init, update, and draw methods in each file
  - Method names must be unique (prepend method with type name)
- Use *extern* to use variables defined in other source files
  - Example: `extern SDL_Surface *screenSurface;`
- Put structure and function prototypes (.h files) so that other files know structure definitions
- Must `#include <SDL.h>` and other libraries in every file that uses them

# Reading Files

- Included with *stdio.h*

```
FILE *myreader = fopen("filename.txt", "r");  
char strLine[64];  
fgets(strLine, 64, myreader);  
fclose(myreader);
```

- Access individual characters in a char \* using array notation
- Add `-mconsole` flag to build options to see output

# Makefiles

```
$ cat Makefile
all:
    gcc -o sdl_shooter sdl_shooter.c enemy.c level_reader.c ship.c `sdl2-config --cflags --libs` -lSDL2_ttf -lSDL2_mixer -mconsole
```

- Can be used to only compile updated files
- Based on dependencies
- Put target at start of line, followed by colon and any dependencies
- Put *tab* character on next line followed by commands
- Use -c parameter to compile C source into object file

```
CC=gcc
EXE=sdl_shooter

$(EXE): sdl_shooter.c enemy.c level_reader.c ship.c
    $(CC) -o $(EXE) sdl_shooter.c enemy.c level_reader.c ship.c `sdl2-config --cflags --libs` -lSDL2_ttf -lSDL2_mixer -mconsole
```

```
gatec@darknut /d/ldsmith/projects/SDLShooter
$ make
gcc -o sdl_shooter sdl_shooter.c enemy.c level_reader.c ship.c `sdl2-config --cflags --libs` -lSDL2_ttf -lSDL2_mixer -mconsole

gatec@darknut /d/ldsmith/projects/SDLShooter
$ make
make: `sdl_shooter' is up to date.
```

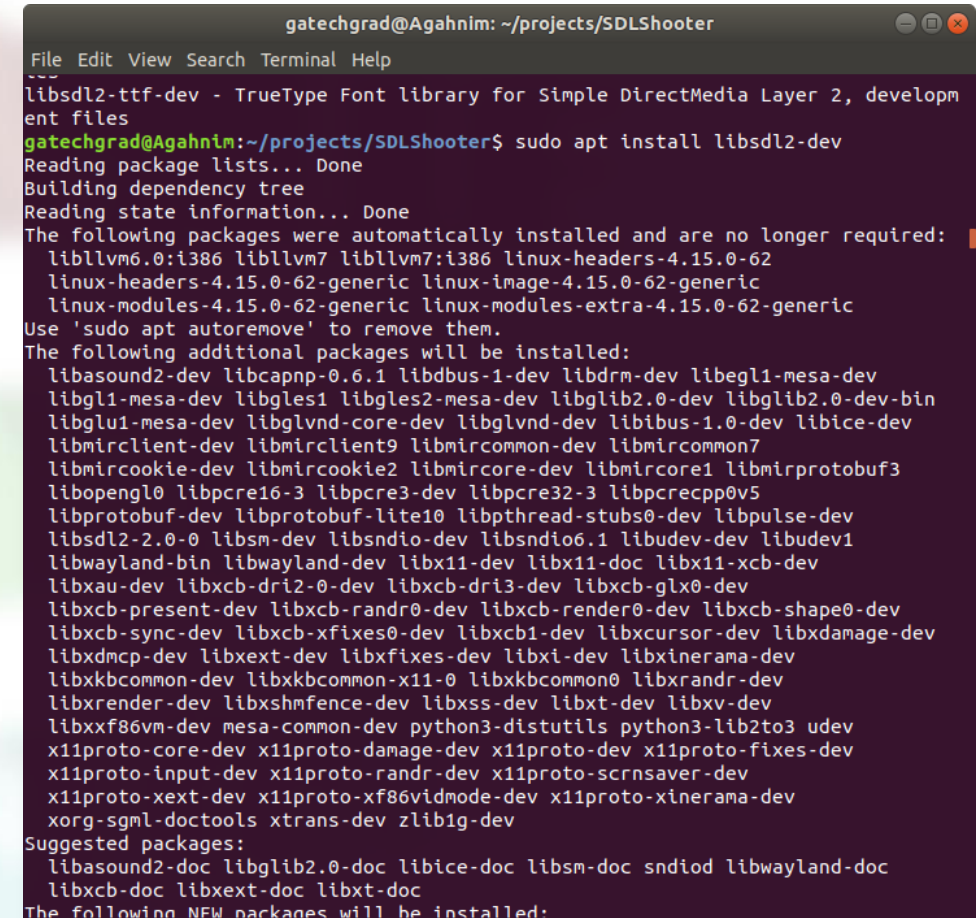
KNOXVILLE  
GAME  
DESIGN



# Building on Linux

- Even easier than Windows
- C compiler (gcc) probably already installed
- Install SDL development libraries with `sudo apt`
- `make linux`

```
$ sudo apt install git
$ git clone https://github.com/gatechgrad/SDLShooter.git
SDLShooter
$ cd SDLShooter
$ apt-cache search ^libsdl # list sdl libraries
$ sudo apt install libsdl2-dev
$ sudo apt install libsdl2-ttf-dev
$ sudo apt install libsdl2-mixer-dev
$ make linux
```



The screenshot shows a terminal window titled "gatechgrad@Agahnim: ~/projects/SDLShooter". The user has run the command `sudo apt install libsdl2-dev`. The terminal output shows the package lists being read, the dependency tree being built, and the state information being read. It then lists packages that were automatically installed and are no longer required, including `libllvm6.0:i386`, `libllvm7:i386`, `linux-headers-4.15.0-62`, `linux-headers-4.15.0-62-generic`, `linux-image-4.15.0-62-generic`, `linux-modules-4.15.0-62-generic`, and `linux-modules-extra-4.15.0-62-generic`. It suggests using `'sudo apt autoremove'` to remove them. Finally, it lists the additional packages that will be installed, including `libasound2-dev`, `libcapnp-0.6.1`, `libdbus-1-dev`, `libdrm-dev`, `libegl1-mesa-dev`, `libgl1-mesa-dev`, `libgles1`, `libgles2-mesa-dev`, `libglu1-mesa-dev`, `libglvnd-core-dev`, `libglvnd-dev`, `libibus-1.0-dev`, `libice-dev`, `libmirclient-dev`, `libmirclient9`, `libmircommon-dev`, `libmircommon7`, `libmircookie-dev`, `libmircookie2`, `libmircore-dev`, `libmircore1`, `libmirprotobuf3`, `libopengl0`, `libpcre16-3`, `libpcre3-dev`, `libpcre32-3`, `libpcrecpp0v5`, `libprotobuf-dev`, `libprotobuf-lite10`, `libpthread-stubs0-dev`, `libpulse-dev`, `libsdl2-2.0-0`, `libsm-dev`, `libsndio-dev`, `libsndio6.1`, `libudev-dev`, `libudev1`, `libwayland-bin`, `libwayland-dev`, `libx11-dev`, `libx11-doc`, `libx11-xcb-dev`, `libxau-dev`, `libxcb-dri2-0-dev`, `libxcb-dri3-dev`, `libxcb-glx0-dev`, `libxcb-present-dev`, `libxcb-randr0-dev`, `libxcb-render0-dev`, `libxcb-shape0-dev`, `libxcb-sync-dev`, `libxcb-xfixes0-dev`, `libxcb1-dev`, `libxcursor-dev`, `libxdamage-dev`, `libxdmcp-dev`, `libxext-dev`, `libxfixes-dev`, `libxi-dev`, `libxinerama-dev`, `libxkbcommon-dev`, `libxkbcommon-x11-0`, `libxkbcommon0`, `libxrandr-dev`, `libxrender-dev`, `libxshmfence-dev`, `libxss-dev`, `libxt-dev`, `libxv-dev`, `libxxf86vm-dev`, `mesa-common-dev`, `python3-distutils`, `python3-lib2to3`, `udev`, `x11proto-core-dev`, `x11proto-damage-dev`, `x11proto-dev`, `x11proto-fixes-dev`, `x11proto-input-dev`, `x11proto-randr-dev`, `x11proto-scrnsaver-dev`, `x11proto-xext-dev`, `x11proto-xf86vidmode-dev`, `x11proto-xinerama-dev`, `xorg-sgml-doctools`, `xtrans-dev`, and `zlib1g-dev`. It also lists suggested packages: `libasound2-doc`, `libgl1-mesa-doc`, `libice-doc`, `libsm-doc`, `sndiod`, `libwayland-doc`, `libxcb-doc`, `libxext-doc`, and `libxt-doc`. Finally, it states that the following NEW packages will be installed.

KNOXVILLE  
GAME  
DESIGN

# Random Numbers

- Seed random number generator with `srand(time(NULL))` ;
  - Must `#include <stdlib.h>` and `<time.h>`
- Call `rand()` ; to get the next random integer
- Use modulo operator ( `%` ) to constrain the range
- Use add ( `+` ) set set minimum value
- No truly random
  - Repeated runs over an small range will return similar values
  - `time(NULL)` only changes once per second

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void) {
    time_t t;
    int iHiddenNumber;
    int iGuess;
    int iGuesses;

    srand(time(NULL));
    iHiddenNumber = rand();
    iHiddenNumber = 1 + (iHiddenNumber % 100);

    iGuesses = 0;
    iGuess = -1;
    while (iGuess != iHiddenNumber) {
        printf("Guess the number between 1 and 100\n");
        scanf("%d", &iGuess);
        iGuesses++;
        if (iGuess > iHiddenNumber) {
            printf("LOWER\n");
        } else if (iGuess < iHiddenNumber) {
            printf("HIGHER\n");
        } else if (iGuess == iHiddenNumber) {
            printf("CORRECT! %d total guesses.\n", iGuesses);
        }
    }
    return 0;
}
```

```
$ ./numguess.exe
Guess the number between 1 and 100
50
LOWER
Guess the number between 1 and 100
25
LOWER
Guess the number between 1 and 100
12
HIGHER
Guess the number between 1 and 100
18
HIGHER
Guess the number between 1 and 100
21
LOWER
Guess the number between 1 and 100
19
HIGHER
Guess the number between 1 and 100
20
CORRECT! 7 total guesses.
```

# More Information

- SDL Forums (active) - <https://discourse.libsdl.org/>
- Documentation Wiki - <http://wiki.libsdl.org/FrontPage>
  - API - <http://wiki.libsdl.org/CategoryAPI>

# Tutorials

- <http://lazyfoo.net/tutorials/SDL/index.php>
- [https://www.tutorialspoint.com/cprogramming/c\\_structures.htm](https://www.tutorialspoint.com/cprogramming/c_structures.htm)
- [sdl.tutorials.com](http://sdl.tutorials.com)
- Game Development with SDL 2.0 -  
<https://www.youtube.com/watch?v=MeMPCSqQ-34>
- <https://www.willusher.io/pages/sdl2/>
- Linked Lists
  - [https://www.learn-c.org/en/Linked\\_lists](https://www.learn-c.org/en/Linked_lists)
  - <http://cslibrary.stanford.edu/103/LinkedListBasics.pdf>