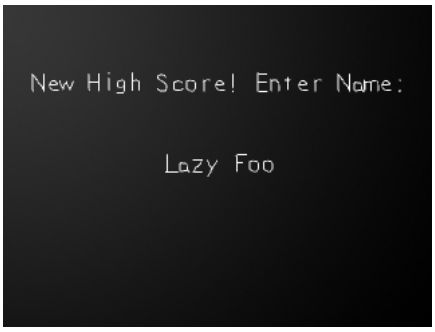
Getting String Input



Last Updated 3/28/10

Here we need to get the player's name after they got a high score. Back in your early days, you used cin to get string input. Since SDL has no built in ways of getting the user's name, we'll make our own function to handle keypresses to get string input.

```
//The key press interpreter
class StringInput
{
    private:
    //The storage string
    std::string str;

    //The text surface
    SDL_Surface *text;

    public:
    //Initializes variables
    StringInput();

    //Does clean up
    ~StringInput();

    //Handles input
    void handle_input();

    //Shows the message on screen
    void show_centered();
};
```

Here's is our class used to manage string input.

In terms of variables we have a string for the actual string data and a surface to render the text to.

Then we have our constructors and destructors, the function handle_input() to deal with user input, and show_centered() to make the string appear on the screen.

```
StringInput::StringInput()
{
    //Initialize the string
    str = "";

    //Initialize the surface
    text = NULL;

    //Enable Unicode
    SDL_EnableUNICODE( SDL_ENABLE );
}
```

```
StringInput::~StringInput()
{
    //Free text surface
    SDL_FreeSurface( text );

    //Disable Unicode
    SDL_EnableUNICODE( SDL_DISABLE );
}
```

In the constructor along with a variable initialization, we enable unicode with SDL_EnableUNICODE(). This will make getting string input much easier as you'll see later.

In the destructor we free our text surface and disable unicode. While enabling unicode makes string input easier, it does add a bit of overhead. It should be turned off when you're not using it.

```
void StringInput::handle_input()
{
    //If a key was pressed
    if( event.type == SDL_KEYDOWN )
    {
        //Keep a copy of the current version of the string
        std::string temp = str;

        //If the string less than maximum size
        if( str.length() <= 16 )
        {
```

Now it's time to handle the user's input.

When the user presses a key, we first store a copy of the current string. I'll tell you why later.

Then we check that the string isn't at maximum length. Here I set it as 16, but you can set it to be whatever you want.

```
            //If the key is a space
            if( event.key.keysym.unicode == (Uint16)' ' )
            {
                //Append the character
                str += (char)event.key.keysym.unicode;
            }
```

The basic concept of string input is when the user presses 'A' add on a 'A', when the user presses 'B' add on a 'B', etc, etc.

Because the SDLKey definitions don't match up with their ASCII/Unicode values, we enable unicode so the "unicode" member of the Keysym structure matches the unicode value of character pressed. Enabling unicode also automatically handles shift and caps lock when you want capital letters and symbols.

If you don't know what unicode is, it's just an extension of ASCII. Instead of being 8bit, it's 16bit so it can hold all the international characters.

Here if the key pressed has the unicode value of the space character, we append it the string. Since a standard string only uses 8bit ASCII characters, we have to convert it to a char when appending it.

```
            //If the key is a number
            else if( ( event.key.keysym.unicode >= (Uint16)'0' ) && ( event.key.keysym.unicode <= (
            {
                //Append the character
                str += (char)event.key.keysym.unicode;
            }
            //If the key is a uppercase letter
            else if( ( event.key.keysym.unicode >= (Uint16)'A' ) && ( event.key.keysym.unicode <= (
            {
                //Append the character
                str += (char)event.key.keysym.unicode;
            }
            //If the key is a lowercase letter
            else if( ( event.key.keysym.unicode >= (Uint16)'a' ) && ( event.key.keysym.unicode <= (
            {
                //Append the character
                str += (char)event.key.keysym.unicode;
            }
```

```
        }
```

In this program we only want spaces (ASCII/Unicode 32), numbers (48-57), uppercase (65-90), and lowercase (97-122) letters to appear. So here we limit the numbers allowed to be appended to the string.

```
        //If backspace was pressed and the string isn't blank
        if( ( event.key.keysym.sym == SDLK_BACKSPACE ) && ( str.length() != 0 ) )
        {
            //Remove a character from the end
            str.erase( str.length() - 1 );
        }
```

Here we deal with when the user presses backspace.

We simply check if the string is empty, and if it's not we lop off the last character of the string.

```
        //If the string was changed
        if( str != temp )
        {
            //Free the old surface
            SDL_FreeSurface( text );

            //Render a new text surface
            text = TTF_RenderText_Solid( font, str.c_str(), textColor );
        }
    }
}
```

Lastly, we check if string was altered by comparing it to the copy we made earlier.

If the string has changed, we free the old text surface, and render a new one.

```
void StringInput::show_centered()
{
    //If the surface isn't blank
    if( text != NULL )
    {
        //Show the name
        apply_surface( ( SCREEN_WIDTH - text->w ) / 2, ( SCREEN_HEIGHT - text->h ) / 2, text, scree
    }
}
```

In our show_centered() function we apply the text surface centered on the screen.

In this program we check if the name surface is NULL because when you try to render a surface from a blank string (that-> ""), SDL_ttf returns NULL. Which makes sense because it was given nothing to render.

```
int main( int argc, char* args[] )
{
    //Quit flag
    bool quit = false;

    //Keep track if whether or not the user has entered their name
    bool nameEntered = false;

    //Initialize
    if( init() == false )
    {
        return 1;
    }

    //The gets the user's name
    StringInput name;

    //Load the files
    if( load_files() == false )
    {
        return 1;
    }

    //Set the message
    message = TTF_RenderText_Solid( font, "New High Score! Enter Name:", textColor );
```

At the top of our main() function we have two new variables. "nameEntered" is a flag

that tells whether or not the user has entered their name which we obviously initialize to false. "name" is, of course, an object of the class we made used to get the user's name.

We have our typical initialization and loading but we also render the message surface before we go into the main loop.

```
    //While the user hasn't quit
    while( quit == false )
    {
        //While there's events to handle
        while( SDL_PollEvent( &event ) )
        {
            //If the user hasn't entered their name yet
            if( nameEntered == false )
            {
                //Get user input
                name.handle_input();

                //If the enter key was pressed
                if( ( event.type == SDL_KEYDOWN ) && ( event.key.keysym.sym == SDLK_RETURN ) )
                {
                    //Change the flag
                    nameEntered = true;

                    //Free the old message surface
                    SDL_FreeSurface( message );

                    //Change the message
                    message = TTF_RenderText_Solid( font, "Rank: 1st", textColor );
                }
            }

            //If the user has Xed out the window
            if( event.type == SDL_QUIT )
            {
                //Quit the program
                quit = true;
            }
        }
```

Here's the event handling part of our main loop.

First we check if the user is still entering their name. If they are we call handle_input() on our StringInput object and let it do its thing.

When the user presses enter, it means the user has finished so we set the "nameEntered" flag to true. Then we free the old message surface and render a new one.

and of course we also check if the user wants to X out.

```
        //Apply the background
        apply_surface( 0, 0, background, screen );

        //Show the message
        apply_surface( ( SCREEN_WIDTH - message->w ) / 2, ( ( SCREEN_HEIGHT / 2 ) - message->h ) /

        //Show the name on the screen
        name.show_centered();

        //Update the screen
        if( SDL_Flip( screen ) == -1 )
        {
            return 1;
        }
    }
```

Now here's the rendering part of our main loop.

Nothing really new here, we just apply the background, then message surface and show our text input.

In this tutorial we only handled string input, but getting integers isn't much harder. The string header has the function atoi() that gets an integer from a string. There's other functions that'll do floating point numbers. Look 'em up.

Download the media and source code for this tutorial here.

Previous Tutorial Next Tutorial

SDL Forums SDL Tutorials Articles OpenGL Tutorials OpenGL Forums
News FAQs Games Contact Donations