Documentation

ahmad-PH edited this page Apr 15, 2019 · 41 revisions

Jump to bottom

`Point`

A struct to store points and coordinates

`Point`

```
Point(int x, int y);
```

Create a Point object. Please note that the y coordinate has a different behaviour in the computer world, than in the math world. The y coordinates grows downwards in computer graphics; Meaning that if a point has a higher y, it will be drawn lower in the screen.

Example:

```
Point P(10, 20);
```

`Rectangle`

A struct representing a rectangle, mainly used to specify where images must be drawn.

`Rectangle`

```
Rectangle(int x, int y, int w, int h)
```

Constructs a rectangle with (x,y) as its top-left corner, w as its width, and h as its height.

```
Rectangle(Point top_left, int w, int h)
```

Constructs a rectangle with `top_left` as its top-left corner, w as its width, and h as its height. handy in case you already have a Point representing the top-left of the rectangle.

```
Rectangle(Point top_left, Point bottom_right)
```

Pretty self-explanatory! handy if you already have the two Points and don't want to unpack them.

RGB

A struct to store colors in Red-Green-Blue format

RGB

```
RGB(int r, int g, int b)
```

Create an RGB object.

Example:

```
RGB PERSIAN_BLUE(28, 57, 187);
```

Window

Window

```
Window(int width = 640, int height = 480, string title = "RSDL")
```

Creates a window with the given width, height, and title.

draw_img

```
void draw_img(string filename, Rectangle dst = NULL_RECT, Rectangle src = NULL_RECT, double angle = 0, boo
```

Draws an image with the specified properties. `dst` specifies the part of the window on which the image will be drawn. `src` specifies the part of the image that you want cropped-out and drawn. If any of these Rectangles are the NULL_RECT, then the entire window/image will be selected. So if `dst` is NULL_RECT, then the image will be drawn over the entire window, and if `src` is NULL_RECT, then all of the image is selected (nothing is cropped out). `angle` is how much the picture must be rotated clockwise. The unit of measurement is degrees. `flip_horizontal` and `flip_vertical` indicate whether the image must be flipped horizontally or vertically, as their name suggests.

Example:

```
win.draw_bmp("myimage.bmp", Rectangle(0,0,200,200));
```

This will draw "my_image.bmp", on the top-left of the window in a rectangle of 200 width and height.

show_text

```
void show_text(string input, Point src, RGB color = WHITE, string font_addr= "FreeSans.ttf", int size = 24
```

Shows the *input* string starting from `src` with the specified color, font and size. `show_text` default font is `FreeSans.ttf` and it can be downloaded from here. Any font used by this function should either be in the same directory where your executable is, or its full address must be passed in as the `font_addr` parameter.

Example:

```
win.show_text("Hello", Point(100, 100), RED, "FreeSans.ttf", 14);
```

`draw_point`

```
void draw_point(Point p, RGB color = WHITE);
```

Draws a point with the specified color.

Example:

```
win.draw_point(Point(100, 100), BLUE);
```

`draw_line`

```
void draw_line(Point src, Point dst, RGB color = WHITE);
```

Draws a line from point `src` to point `dst` with the specified color.

Example:

```
win.draw_line(Point(0, 0), Point(100, 100), RED);
```

`draw_rect`

```
void draw_rect(Rectangle rect, RGB color = WHITE, unsigned int line_width = 4);
```

Draws a rectangle with the specified line_width and color. This will only draw the borders of the rectangle, i.e. the rectangle will be hollow. To draw a filled rectangle, use `fill_rect`.

Example:

```
win.draw_rect(Rectangle(100, 100, 50, 50), BLUE);
```

`fill_rect`

```
void fill_rect(Rectangle rect, RGB color = WHITE);
```

Draws a filled rectangle with the specified color.

Example:

```
win.fill_rect(Point(100, 100), Point(50, 100), BLUE);
```

`fill_circle`

```
void fill_circle(Point center, int radius, RGB color);
```

Draws a filled circle with the specified color, center and radius.

Example:

```
win.fill_circle(Point(100, 100), 10, BLUE);
```

update_screen

```
void update_screen();
```

Updates the window to show the effect of all the methods that have visually manipulated the window. Call this method after all of your calls to draw, fill, or show methods to update the screen.

Example:

```
win.update_screen();
```

clear

```
void clear();
```

Clears the entire screen.

Example:

```
win.clear();
```

get_width

```
int get_width();
```

Returns the width of the window.

get_height

```
int get_height();
```

Returns the height of the window.

play_sound_effect

```
void play_sound_effect(string filename);
```

Plays the music file with name `filename`. This is only suitable for short-duration files, a.k.a. sound effects. the only supported format is .wav. This function does not interfere with `play_music`. So you can have as many calls to `play_sound_effect` without disrupting the music which is played using `play_music`.

play_music

```
void play_music(string filename);
```

Plays the music file with name `filename`. This is suitable for background music of long durations. Only one music can be played at a time, and if the function is called twice, the first music will be stopped before playing the new one. If a music with the

same filename was paused using `pause_music`, it will resume playing the music. If you want the music to start playing from the beginning, you should use `stop_music` first.

`pause_music`

```
void pause_music();
```

Pauses the currently playing music, if there is any.

`resume_music`

```
void resume_music();
```

Resumes the currently paused music, if there is any.

`stop_music()`

```
void stop_music();
```

Stops the current music. Calling `play_music` with the filename it was called with the last time, will play the music from the beginning.

`has_pending_event`

```
bool has_pending_event();
```

Checks whether or not there are any unprocessed events waiting in the event queue. Returns true if there are.

```
bool pending_event_exists = win.has_pending_event();
```

`poll_for_event`

```
Event poll_for_event();
```

Returns the last event that has occured in window, and removes it from the event queue. If no events have occured since the last call to `poll_for_event`, an event will be returned whose `Event::get_type()` method will return `NA`. But this is not a reliable behavior and it is recommended that you check for the existence of events using `Window::has_pending_event()`.

Example:

```
Event last_event = win.poll_for_event();
```

`Event`

Event objects are returned by window *poll_for_event* method
You can get event data or event type from event.

`get_type`

```
EventType get_type();
```

Returns type of event.

It can be one the following types:

- `NA` : Returned when no event has occured in the window since the last call to`poll_for_event` .
- `LCLICK` : Returned when mouse left button was clicked.
- `RCLICK` : Returned when mouse right button was clicked.
- `LRELEASE` : Returned when mouse left button is released.
- `RRELEASE` : Returned when mouse right button is released.
- `MMOTION` : Returned when the mouse cursor moves.
- `KEY_PRESS` : Returned when a keyboard key was pressed.
- `KEY_RELEASE` : Returned when a keyboard key was released.
- `QUIT` : Returned when a quit from program command was issued.

```
Event::EventType type = get_type();
switch(type) {
    case Event::LCLICK:
        //handle left-click
        break;
    .
    .
    .
}
```

you can use the methods introduced so far to perform event handling like this:

```
while(win.has_pending_event()) {
    Event event = win.poll_for_event();
    switch(event.get_type()) {
        case Event::LCLICK:
            //handle left-click
            break;
        .
        .
        .
    }

}
```

`get_mouse_position`

```
Point get_mouse_position();
```

Returns the mouse position in window, at the time of the event.
Returns relative valuse if event is a mouse motion.
Returns `-1` if event is a keyboard key press.

Example:

```
int mouse_y_location;
if(event.get_type() == Event::LCLICK )
   Point p = event.get_mouse_position();
```

`get_pressed_key`

```
char get_pressed_key();
```

Returns the character value associated with the pressed keyboard key. This method will have undefined behavior if the

pressed key does not have a corresponding ASCII character (for example, the arrow keys). Returns `-1` if event is not a keyboard key press.

Example:

```
char pressedChar;
if(event.get_type() == Event::KEY_PRESS ) {
   pressed_char = event.get_pressed_key();
   if (pressed_char == 'w') {
      //handle pressing of the 'w' key
   }
   .
   .
   .
}
```

Miscellaneous

`get_current_mouse_position`

```
Point get_current_mouse_position();
```

Returns the current position of the mouse.

`delay`

```
void delay(int milliseconds);
```

stops the execution of the programs for a duration of `milliseconds` milliseconds.

Example:

```
while (...) {
    win.draw_img("next_frame_of_animation");
    delay(40);  // the animation will have 25fps
}
```