Ca’ Foscari
Introduzione Alla Programmazione

M. De Zorzi
December 3, 2021

libdz
Input management

Preamble

The classic approach

```
while (1) {
  scanf("%c", c); // blocks execution
  handle_input(c);
  update_field();
  print_field();
}
```

What we're looking for

+ Call function on input without blocking the the whole program (Possibly on a keypress, instad of on **enter**).
+ Update screen every **x** seconds.
+ Update the field every **y** seconds.

Motivation: Tetris redraws the screen 29.97 times a second. At level 1 blocks move every 48 frames, at level 20 every single frame!

A solution

```
term_init();
while (1) {
  c = select_char(); // doesn't block!
                     // returns KEY_NONE if nothing is pressed
  if (c != KEY_NONE) {
    handle_input(c);
  }
  if (now() - update_time > 50) {
    update_field();
    update_time = now();
  }
  if (now() - draw_time > 100) {
    print_field();
    draw_time = now();
  }
}
term_end();
```

select_char implementation

```
char select_char()
{
  char buff;
  int r = read(STDIN_FILENO, &buff, 1);
  if (r > 0) {
    return buff;
  } else {
    return 0;
  }
}
```

A more elegant select_char implementation

```
char select_char()
{
  char buff;
  return read(STDIN_FILENO, &buff, 1) > 0 ? buff : 0;
}
```

term_init implementation

```
#define ANSI_CLEAR_SCREEN "\033[1;1H\033[2J"
#define ANSI_HIDE_CURSOR  "\033[?25l"
#define ANSI_SHOW_CURSOR  "\033[?25h"
```

term_init implementation

```
void term_init() {
  struct termios new_settings;

  printf(ANSI_CLEAR_SCREEN);
  printf(ANSI_HIDE_CURSOR);

  tcgetattr(0, &term_settings);
  new_settings = term_settings;
  new_settings.c_lflag &= ~ICANON;
  new_settings.c_lflag &= ~ECHO;
  new_settings.c_lflag &= ~ISIG;
  new_settings.c_cc[VMIN]  = 0;
  new_settings.c_cc[VTIME] = 0;
  tcsetattr(0, TCSANOW, &new_settings);
}
```

end_term implementation

```
void term_end() {
  printf(ANSI_CLEAR_SCREEN);
  printf(ANSI_SHOW_CURSOR);

  tcsetattr(0, TCSANOW, &term_settings);
}
```

What about mouse support?

A better solution

```
typedef struct {
  int x;
  int y;

  enum {
    MOUSE_PRESS   = 0,
    MOUSE_RELEASE = 1,
    MOUSE_MOVE    = 35,
  } type;

  enum {
    BUTTON_LEFT   = 0,
    BUTTON_MIDDLE = 1,
    BUTTON_RIGHT  = 2,
  } button;
} mouse_event_t;
```

```
typedef struct {
  int key;
  enum {
    MOD_NONE  = 0,
    MOD_CTRL  = 1,
    MOD_SHIFT = 2,
  } modifier;
} kb_event_t;
```

```
typedef struct {
  union {
    kb_event_t    kb;
    mouse_event_t mouse;
  } event;
  enum {
    EV_KEYBORAD,
    EV_MOUSE,
    EV_NONE,
  } type;
} input_event_t;
```

init_term fix

```
void term_init() {
  // ...
  // enables mouse reporting
  printf("\033[?1000;1002l\033[?1003;1006h");
}
```

Special Control Sequences

```
# Keyboard
\033[A # up arrow
\033[B # down arrow
\033[C # left arrow
\033[D # right arrow

# Mouse
0;24;27M # left click, at position (24, 27), pressed
2;19;30m # right click, at position (19, 30), released
```

Input parsing

```
input_event_t parse_input() {
  char data[256] = {0};
  int index = 0, len = 0;
  while ((data[index++] = select_char()) && index < 256);
  len = strlen(data);
  if (len > 0) {
    if (len > 1 && (data[len - 1] == 'm' || data[len - 1] == 'M')) {
      return (input_event_t){
        .event = { .mouse = parse_mouse(data) },
        .type = EV_MOUSE };
    } else {
      return (input_event_t){
        .event = { .kb = parse_key(data) },
        .type = EV_KEYBORAD };
    }
  }
  return (input_event_t){ .type = EV_NONE, };
}
```

Keyboard parsing (1)

```
#define CTRLMASK(Chr)  ((Chr)&0x1f)
#define SHIFTMASK(Chr) ((Chr)|0x40)

kb_event_t parse_key(char *data) {
  char c = data[0];
  switch (c) {
  case 0: return (kb_event_t){.key = KEY_NONE};
  case KEY_ESCAPE:
    if (data[1] == '[') {
      switch (data[2]) {
      case 'A': return (kb_event_t){.key = KEY_UP};
      case 'B': return (kb_event_t){.key = KEY_DOWN};
      case 'C': return (kb_event_t){.key = KEY_LEFT};
      case 'D': return (kb_event_t){.key = KEY_RIGHT};
      default: return (kb_event_t){.key = KEY_NONE};
      }
    }
    return (kb_event_t){.key = KEY_NONE};
    // ...
```

Keyboard parsing (2)

```
    // ...
    default:
      return (kb_event_t){
          .key = tolower(c),
          .modifier = (c == CTRLMASK(c) ? MOD_CTRL : MOD_NONE)
                    | (c != SHIFTMASK(c) ? MOD_SHIFT : MOD_NONE),
      };
    }
  }
```

Mouse parsing

```
mouse_event_t parse_mouse(char *data) {
  int x, y;
  unsigned button;
  char c;

  sscanf(data + 3, "%d;%d;%d%c", &button, &at.x, &at.y, &c);

  x -= 1;
  y -= 1;

  return (mouse_event_t){
      .x      = x,
      .y      = y,
      .type   = button == 35 ? MOUSE_MOVE : c == 'M',
      .button = button,
  };
}
```

```
while (e->run) {
  ev = parse_input();

  screen_update(e->screen);

  if (ev.type == EV_KEYBORAD) {
    (e->hook_kb)(e, ev.event.kb);
  } else if (ev.type == EV_MOUSE) {
    (e->hook_mouse)(e, ev.event.mouse);
  }
  // ...
  // ...
  if (e->run) {
    (e->hook_loop)(e, dt);
  }

  if (current() - 1e5 > e->last_draw) {
    screen_render(e->screen);
    e->last_draw = current();
  }

  screen_repaint(e->screen);

  usleep(1e8 / dt);
}
```

```
void
hook_keyboard(void *ptr, kb_event_t ev)
{
  engine_t *e = ptr;

  switch (ev.key) {
    case 'q': e->run = false; break;
    case /* ... */: /* ... */
  }
}
```

```
void
hook_mouse(void *ptr, mouse_event_t ev)
{
  engine_t *e = ptr;

  statusbar_fupdate(sb, "%d:%d",
                    ev.at.x, ev.at.y);
}
```

References

+ invisible-island.net/xterm/ctlseqs/ctlseqs.html
+ https://viewsourcecode.org/snaptoken/kilo/02.enteringRawMode.html
+ github.com/mircodezorzi/libdz/blob/master/src/term/mouse.c
+ github.com/mircodezorzi/libdz/blob/master/src/term/keyboard.c
+ github.com/mircodezorzi/libdz/blob/master/src/term/input.c
+ Want unicode support in ansi C?
  https://github.com/mircodezorzi/libdz/blob/master/src/utf8.c

Questions?