

Computer Labs: Lab 5 & Sprites

2º MIEIC

Pedro F. Souto (`pfs@fe.up.pt`)

April 12, 2024

The “Class” Sprite: `sprite.h` (by jcard@fe.up.pt)

Sprite “Two-dimensional image that is integrated into a larger scene” (Wikipedia)

- ▶ Allows the integration of independent pixmaps into a scene
- ▶ Allows image animation without altering the background – thus a sprite can be considered an overlay image

```
typedef struct {  
    int x, y; // current position  
    int width, height; // dimensions  
    int xspeed, yspeed; // current speed  
    char *map; // the pixmap  
} Sprite;
```

The pixmap uses **black** (or some unused color) for the background, which is assumed to be transparent

The “Class” Sprite: sprite.c (1/2)

```
/** Creates a new sprite from XPM "pic", with specified
 * position (within the screen limits) and speed;
 * Does not draw the sprite on the screen
 * Returns NULL on invalid pixmap.
 */
Sprite *create_sprite(const char *pic[], int x, int y,
                      int xspeed, int yspeed) {
    //allocate space for the "object"
    Sprite *sp = (Sprite *) malloc ( sizeof(Sprite));
    xpm_image_t img;
    if( sp == NULL )
        return NULL;
    // read the sprite pixmap
    sp->map = xpm_load(pic, XPM_INDEXED, &img);
    if( sp->map == NULL ) {
        free(sp);
        return NULL;
    }
    sp->width = img.width; sp->height=img.height;
    ...
    return sp;
}
```

The “Class” Sprite: `sprite.c` (2/2)

```
void destroy_sprite(Sprite *sp) {
    if( sp == NULL )
        return;
    if( sp ->map )
        free(sp->map);
    free(sp);
    sp = NULL;          // XXX: pointer is passed by value
                        //      should do this @ the caller
}

// XXX: move_sprite would be a more appropriate name
int animate_sprite(Sprite *sp) {
    ...
}

/* Some useful non-visible functions */
static int draw_sprite(Sprite *sp, char *base) {
    ...
}
static int check_collision(Sprite *sp, char *base) {
    ...
}
```

Lab 5: test_move() (again)

```
int video_test_move( const char *xpm[],,  
                    uint16_t xi, uint16_t yi,  
                    uint16_t xf, uint16_t yf  
                    int8_t speed, uint8_t frame_rate)
```

What? Move a sprite on the screen (only along the x or y axes)

`xpm` XPM for the sprite

`(xi, yi)` initial coordinates (of ULC)

`(xf, yf)` final coordinates (of ULC)

`speed` speed

If **non-negative** number of pixels between consecutive frames

If **negative** number of frames required for a 1 pixel movement

`frame_rate` number of frames per second

How? Should use the `sprite` "class"

- ▶ But you can change it slightly (I did).
- ▶ Need not implement all functions.

Sprite Animation

- ▶ Animation of a sprite can be achieved by presenting a sequence of pixmaps
 - ▶ Each pixmap (but the first) in this sequence differs slightly from the previous pixmap



- ▶ To create an animated sprite we need to specify several pixmaps
 - ▶ This can be done in different ways
 - ▶ E.g. using an array of pixmaps
- ▶ We'll use a C function with a variable number of arguments:
`AnimSprite *create_animSprite(uint8_t no_pic, char *pic1[], ...);`
`printf()` is the most common C function of this type

(Functions with a Variable Number of Arguments (1/2))

- ▶ Must have at least one named argument
 - ▶ The unnamed arguments are passed in a list
- ▶ But, needs to know
 - ▶ How many arguments in the list
 - ▶ The type of each of these arguments

both of which can vary from one invocation to the next

- ▶ Uses a kind of an iterator, of type `va_list`, to traverse the list
- ▶ Relies on a set of macros defined in `<stdarg.h>`, whose first parameter is the "iterator":

`va_start` initializes the iterator to the first argument of the list

- ▶ `va_start` second argument is the **last known function argument**

`va_arg` to get the next argument of the list, i.e. to iterate through the list

- ▶ `va_arg` second argument is the type of that unnamed argument
- ▶ On first invocation, returns the first argument after the last known argument, i.e. the first unnamed argument in the list

`va_end` to finalize the access

(Functions with a Variable Number of Arguments (2/2))

```
#include <stdarg.h> // va_* macros are defined here
int foo(int required, ...) {
    va_list ap; // "pointer" to next argument
    va_start(ap, required); // initializes ap to point to
        // first (unnamed) argument of the list ;
        // the 2nd argument of va_start is the last named argument
    int i = va_arg(ap, int); // accesses the next list argument
        // the second argument of va_arg() is the type
        // on first call, returns the value of the first
        // argument after the last fixed argument
    float f = va_arg(ap, float);
    char *s = va_arg(ap, char *);
    va_end(ap); // must be called to finalize the access
}
```

Question How do you know the type of each argument of the list?

Answer There are several possibilities:

`printf()` uses the format string

`create_animSprite()` all arguments have the same known type

The “Class” Animated Sprite: AnimSprite.h

```
#include <stdarg.h> // va_* macros are defined here
#include "sprite.h"
typedef struct {
    Sprite *sp;           // standard sprite
    int aspeed;           // no. frames per pixmap
    int cur_aspeed;       // no. frames left to next change
    int num_fig;          // number of pixmaps
    int cur_fig;          // current pixmap
    char **map;           // array of pointers to pixmaps
} AnimSprite;

create_animSprite(uint8_t no_pic, char *pic1[], ...);
int animate_animSprite(AnimSprite *sp,);
void destroy_animSprite(AnimSprite *sp);
```

Animation speed is measured as number of “frames” per pixmap, i.e. the number of frames each pixmap is displayed in an animation.

- The smaller this value, the faster the animation

The “Class” Animated Sprite: AnimSprite.c (1/2)

```
AnimSprite *create_animSprite(uint8_t no_pic,  
                               char *pic1[], ...) {  
    AnimSprite *asp = malloc(sizeof(AnimSprite));  
    // create a standard sprite with first pixmap  
    asp->sp = create_sprite(pic1,0,0,0,0);  
    // allocate array of pointers to pixmaps  
    asp->map = malloc((no_pic) * sizeof(char *));  
    // initialize the first pixmap  
    asp->map[0] = asp->sp->map;  
    // continues in next transparency
```

The “Class” Animated Sprite: AnimSprite.c (2/2)

```
// initialize the remainder with the variable arguments
// iterate over the list of arguments
va_list ap;
va_start(ap, pic1);
for( i = 1; i <no_pic; i++ ) {
    char **tmp = va_arg(ap, char **);
    xpm_image_t img;
    asp->map[i] = xpm_load(tmp, XPM_INDEXED, &img);
    if( asp->map[i] == NULL
        || img.width != asp->sp->width || img.height != asp->sp->height )
        // failure: release allocated memory
        for(j = 1; j<i;j++)
            free(asp->map[j]);
    free(asp->map);
    destroy_sprite(asp->sp);
    free(asp);
    va_end(ap);
    return NULL;
}
}
va_end(ap);
...
```

Thanks to:

Based on material by:

► João Cardoso (jcard@fe.up.pt)

Further Reading

- ▶ João Cardoso, [Notas sobre *Sprites*](#)