Technical documentation written in a slightly free manner

What's going on in main()?

The program is started from the function **main()**. It fills the arrays of the displayed game elements using the functions **create_aliens()**, **create_space_ship()**, **create_bullets()**, **create_walls()**, **create_lives()**, **create_game_text()**.

Then **init_model()** is called, which setup memory mapping which provides access to the peripheral registers region of RGB LEDs, knobs and line of yellow LEDs.

Then **init_drawing()** is called, which allocates memory for the array corresponding to the display and setup memory mapping which provides access to the peripheral registers region of LCD display.

After that, an infinite loop **while(true)** is started, which acts as a game loop. The functions **render(objects, OBJECTS_NUM)** and **advance_state(objects, OBJECTS_NUM)** are called in it, which are responsible for rendering and updating the state of all game objects respectively.

How is storage and designation organized for all game objects?

Each element is individually represented as an object of the **object_desc_t** structure, which contains attributes denoting such data as width, height, scale, array denoting the corresponding texture, position on the display, etc.

Then these elements depending on what they are responsible for are combined into the **objects_t** structure which contains an array of objects_desc_t, their number, index of the current object, color and speed.

Both structures can be found in the file object_structure.h.

An array of objects_t structure is created and populated directly in the main() for each type of game element (aliens, player's spaceship, text, bullets, walls, etc.)

Textures

To determine the graphic forms of individual characters, numbers, symbols directly, it was decided to use matrices with 32-bit binary numbers.

For each type of game elements there is a static array of bits with binary written numbers and arrays that describe the width and height of each matrix in the array. In fact, it helps to separate matrices of different objects as they are written in a row.

Such arrays are:

- **aliens_bits[]** and its height and width arrays - **aliens_height[]**, **aliens_width[]**
- **bullets_bits[]**
- **char_bits[]** - for font (and **char_height[]**, **char_width[]**)
- **space_ship_bits[]**
- **wall_pieces_bits[]**

Some basic classes and their some basic functions:

model.c

The model.c file contains functions connected with game logic. For example, function **advance_state(objects_t** objects, int obj_num)**, called in main.c. This function calls functions from game_object files connected with changes of game object coordinates (**move_aliens(objects_t* aliens)**, **void move_space_ship_bullet(object_desc_t* bullet, int pos_x, bool red_knob_pressed)**, **move_flying_saucer(objects_t * flying_saucer_obj)** and others), the intersections of objects are detected **detect_intersections(objects_t** objects)**, the representation on the LED is updated **update_leds(objects_t** objects)** to LED.

Also in model.c the player's input is listened for via rotary dials knobs. The functions responsible for this are: **get_pos_x_from_blue_knob()**, **if_red_knob_pressed()**.

view.c

Contains functions responsible for displaying all game elements on the LCD display. The main function is **render(objects_t** objects, int obj_num)**, called in the main.c. It calls functions such as **draw_background_to_array()** - which draws the background, **draw_objects_to_array(object_desc_t *objects, int count, uint16_t color)** - which passes through all visible game elements and puts them into the array 480 × 320 × 2 indicating the LCD display and then function **draw_objects_to_array(object_desc_t *objects, int count, uint16_t color)**, which transfers the array 480 × 320 × 2 already filled in the display itself (using the functions **parlcd_write_cmd(unsigned char *parlcd_mem_base, uint16_t cmd)** and **parlcd_write_data(unsigned char *parlcd_mem_base, uint16_t data)** from the mzapo_parlcd. c folder kit_tools).

aliens.c

This file contains functions that describe the behavior and properties, as well as their changes to game elements of a particular type - aliens. Other files with game objects have a similar functionality with some changes, taking into account what they are generally needed for.

Contains function **create_aliens()** - which fills array aliens_desc with structures object_desc_t and then creates structure objects_t in which this array is being put. This is where all aliens properties are assigned - dimensions, positions, speed, textures, scale, cost, etc.

There is also function **move_aliens(objects_t* aliens)**, where we pass through an array with aliens, and for those that we have not hit yet, call function **move_alive_alien(objects_t* aliens, int curr_idx)**. Depending on the current position, it changes speed and adds it to the current position, thus moving the object.

There's also the **reset_aliens(objects_t* aliens)** function. It sets all killed aliens back when transitioning to a new game stage (when player has killed all aliens and didn't die himself). It works on the same principle as create_aliens().

A little bit about copyrights.

As the background of the game was used the cover of Pink Floyd's 1973 album The Dark Side Of The Moon.