

Space Invaders

Generated by Doxygen 1.8.15

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 collider Struct Reference	5
3.1.1 Detailed Description	5
3.2 laser Struct Reference	5
3.2.1 Detailed Description	6
3.3 Sprite Struct Reference	6
3.3.1 Detailed Description	6
4 File Documentation	7
4.1 asserts.h File Reference	7
4.1.1 Detailed Description	7
4.2 combat.h File Reference	7
4.2.1 Detailed Description	8
4.2.2 Function Documentation	8
4.2.2.1 animate_lasers()	8
4.2.2.2 register_collide_callback()	8
4.2.2.3 register_collider()	9
4.2.2.4 remove_collider()	9
4.2.2.5 reset_combat()	9
4.2.2.6 shoot_laser()	10
4.3 input.h File Reference	10
4.3.1 Detailed Description	10
4.3.2 Function Documentation	10
4.3.2.1 get_input()	11
4.3.2.2 init_input()	12
4.4 lcd.h File Reference	12
4.4.1 Detailed Description	13
4.4.2 Function Documentation	13
4.4.2.1 clear_page()	13
4.4.2.2 display_on()	13
4.4.2.3 send_data()	14
4.4.2.4 send_instruction()	14
4.4.2.5 set_address()	14
4.4.2.6 set_page()	14
4.5 main.c File Reference	15
4.5.1 Detailed Description	16
4.5.2 Function Documentation	16

4.5.2.1	animatePlayer()	17
4.5.2.2	calculate_bounds()	17
4.5.2.3	drawLives()	17
4.5.2.4	drawScore()	17
4.5.2.5	init_game()	17
4.5.2.6	onCollide()	18
4.5.2.7	player_movement()	18
4.5.2.8	renderGame()	18
4.5.2.9	renderGameOver()	18
4.5.2.10	shoot_monsters()	18
4.5.3	Variable Documentation	19
4.5.3.1	default_monsters	19
4.5.3.2	default_walls	19
4.6	rendering.h File Reference	19
4.6.1	Detailed Description	20
4.6.2	Function Documentation	20
4.6.2.1	fill_rect()	20
4.6.2.2	set_pixel()	20
4.7	sprites.h File Reference	21
4.7.1	Detailed Description	22
4.7.2	Typedef Documentation	22
4.7.2.1	sprite	22
4.7.3	Function Documentation	22
4.7.3.1	draw_sprite()	22
4.7.4	Variable Documentation	23
4.7.4.1	HEART	23
4.7.4.2	LETTER_A	23
4.7.4.3	LETTER_E	23
4.7.4.4	LETTER_G	23
4.7.4.5	LETTER_M	24
4.7.4.6	LETTER_O	24
4.7.4.7	LETTER_R	24
4.7.4.8	LETTER_V	24
4.7.4.9	MONSTER	24
4.7.4.10	MONSTER2	24
4.7.4.11	PLAYER	25
4.7.4.12	WALL	25
4.7.4.13	WALL_DAMAGE_1	25
4.7.4.14	WALL_DAMAGE_2	25
4.7.4.15	WALL_DAMAGE_3	25
4.8	wait.h File Reference	25
4.8.1	Detailed Description	26

4.8.2 Function Documentation	26
4.8.2.1 delay_millis()	26
Index	27

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

collider	Representing an object which can be hit by a laser ray	5
laser	Representing a Laser Ray	5
Sprite	Representing a Sprite which can be drawn onto the display	6

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

adc.h	??
asserts.h		
Asserts	7
combat.h		
Shooting logic	7
input.h		
Handling the Input via ADC	10
lcd.h		
Display driver	12
main.c		
Main	15
rendering.h		
Display API	19
sprites.h		
Sprites for the game	21
sprites_numbers.h	??
wait.h		
Delay	25

Chapter 3

Class Documentation

3.1 collider Struct Reference

Representing an object which can be hit by a laser ray.

```
#include <combat.h>
```

Public Attributes

- `int8_t x`
- `int8_t y`
- `int8_t width`
- `int8_t height`

3.1.1 Detailed Description

Representing an object which can be hit by a laser ray.

Parameters

<i>x</i>	X Coordinate
<i>y</i>	Y Coordinate
<i>width</i>	Width of the object in pixels
<i>height</i>	Height of the object in pixels

The documentation for this struct was generated from the following file:

- [combat.h](#)

3.2 laser Struct Reference

Representing a Laser Ray.

Public Attributes

- `uint8_t x`
- `uint8_t y`
- `int8_t dir`
- `bool deleted`

3.2.1 Detailed Description

Representing a Laser Ray.

The documentation for this struct was generated from the following file:

- `combat.c`

3.3 Sprite Struct Reference

Representing a [Sprite](#) which can be drawn onto the display.

```
#include <sprites.h>
```

Public Attributes

- `uint16_t data` [[MAX_SPRITE_HEIGHT](#)]
- `uint8_t width`
- `uint8_t height`

3.3.1 Detailed Description

Representing a [Sprite](#) which can be drawn onto the display.

Parameters

<i>data</i>	An Array which contains <code>uint16_t</code> variables, matching the pixels of a row
<i>width</i>	The width of a sprite in pixels
<i>height</i>	The height of a sprite in pixels

The documentation for this struct was generated from the following file:

- [sprites.h](#)

Chapter 4

File Documentation

4.1 asserts.h File Reference

Asserts.

```
#include "assert.h"
```

Macros

- `#define ERR_DEFAULT 0b11111111`
- `#define ERR_LOOP_EXIT 0b11110000`
- `#define ERR_INIT 0b10000001`
- `#define ERR_ARRAY_FULL 0b11100111`
- `#define ERR_NULL_POINTER 0b00011000`
- `#define ERR_OVERFLOW 0b00000111`
- `#define ASSERT_LED(code, expr) if(expr == 0){DDRF = 0xFF; PORTF = code; while(1);}`

4.1.1 Detailed Description

Asserts.

4.2 combat.h File Reference

Shooting logic.

```
#include <avr/io.h>
```

Classes

- struct [collider](#)

Representing an object which can be hit by a laser ray.

Macros

- `#define DIRECTION_DOWN 1`
A laser ray is moving downwards (shot by a monster)
- `#define DIRECTION_UP -1`
A laser ray is moving upwards (shot by a monster)

Functions

- `uint8_t register_collider (collider *c)`
Register a new collider.
- `void remove_collider (uint8_t index)`
Removes a given collider.
- `void register_collide_callback (void(*func)(uint8_t))`
Register a function to handle a collision.
- `void shoot_laser (uint8_t x, uint8_t y, int8_t direction)`
Spawn a new laser ray.
- `void animate_lasers (void)`
Animates all currently active laser rays.
- `void render_lasers (void)`
Renders all currently active laser rays.
- `void reset_combat (void)`
Reset everything.

4.2.1 Detailed Description

Shooting logic.

4.2.2 Function Documentation

4.2.2.1 animate_lasers()

```
void animate_lasers (
    void )
```

Animates all currently active laser rays.

Used to track the movement of laser rays.

4.2.2.2 register_collide_callback()

```
void register_collide_callback (
    void(*) (uint8_t) func )
```

Register a function to handle a collision.

To set the function which is executed if a collision is detected

Parameters

<i>func</i>	the function with signature void func(uint8_t)
-------------	---

4.2.2.3 register_collider()

```
uint8_t register_collider (  
    collider * c )
```

Register a new collider.

A collider is used to detect and handle collisions of Lasers and Objects (Monster, Player or Wall).

Parameters

<i>c</i>	A pointer to the collider
----------	---------------------------

Returns

uint8_t the id of the collider that can be used to remove it later

4.2.2.4 remove_collider()

```
void remove_collider (  
    uint8_t index )
```

Removes a given collider.

Parameters

<i>index</i>	the id of the collider
--------------	------------------------

4.2.2.5 reset_combat()

```
void reset_combat (  
    void )
```

Reset everything.

This function is used to reset all variables in order to start a new game.

4.2.2.6 shoot_laser()

```
void shoot_laser (
    uint8_t x,
    uint8_t y,
    int8_t direction )
```

Spawn a new laser ray.

Parameters

<i>x</i>	the x position
<i>y</i>	the y position
<i>direction</i>	the direction it is supposed to move in

4.3 input.h File Reference

Handling the Input via ADC.

```
#include <avr/io.h>
#include "adc.h"
#include <stdbool.h>
```

Macros

- `#define JOY_X 8`
The Pin which is connected to the X-Axis data of the Joystick.
- `#define JOY_Y 9`
The Pin which is connected to the Y-Axis data of the Joystick.

Functions

- void `init_input` (void)
Initialize the Input ADC.
- void `get_input` (int16_t *x, int16_t *y)
Get the x and y input from the definend ADC pin.

4.3.1 Detailed Description

Handling the Input via ADC.

4.3.2 Function Documentation

4.3.2.1 get_input()

```
void get_input (
    int16_t * x,
    int16_t * y )
```

Get the x and y input from the definend ADC pin.

Parameters

<i>x</i>	Input on the x axis
<i>y</i>	Input on the y axis

4.3.2.2 init_input()

```
void init_input (
    void )
```

Initialize the Input ADC.

4.4 Icd.h File Reference

Display driver.

Macros

- #define **LCD_BUSY** 7
- #define **TIMER_DELAY** 10
- #define **OP_DISPLAY_ON** 0b00111111
- #define **TIMER_SIGNAL** ((PORTC >> DDC5) & 1)
- #define **bit**(x) (1 << x)

Functions

- void [clock_toggle](#) (void)
Toggles the clock signal needed for synchronization of the lcd.
- void [set_data_input](#) (void)
set the bus to data mode
- void [set_instruction_input](#) (void)
set the bus to instruction mode
- void [set_data_write](#) (void)
set the data mode to write
- void [set_data_read](#) (void)
set the data mode to read
- void [set_reset](#) (bool val)
set the reset bit
- void [set_data](#) (uint8_t data)
set the data on the data bus
- void [chipselect_1](#) (bool val)
select the first chip controlling the left half of the screen
- void [chipselect_2](#) (bool val)
select the second chip controlling the right half of the screen
- void [chipselect](#) (uint8_t chip)

- select the given chip*
- void `flash_data` (void)
 - flash the data or instruction on the bus to the lcd*
- void `send_instruction` (uint8_t instr)
 - send an instruction to the lcd*
- void `send_data` (uint8_t data)
 - send data to the lcd*
- void `display_on` (void)
 - turn the display on*
- void `set_address` (uint8_t addr)
 - set the column in the current page*
- void `set_page` (uint8_t page)
 - select the page*
- void `clear_page` (uint8_t page)
 - clears the given page*
- void `clear_screen` (void)
 - clears the whole screen*

4.4.1 Detailed Description

Display driver.

4.4.2 Function Documentation

4.4.2.1 `clear_page()`

```
void clear_page (
    uint8_t page )
```

clears the given page

Parameters

<i>page</i>	the numer of the page, from 0 to 7
-------------	------------------------------------

4.4.2.2 `display_on()`

```
void display_on (
    void )
```

turn the display on

Note, that this will only turn on selected chips, so in order to turn on the whole screen, call `chipselect_1(true)` and `chipselect_2(true)` beforehand

4.4.2.3 send_data()

```
void send_data (
    uint8_t data )
```

send data to the lcd

Parameters

<i>data</i>	the data
-------------	----------

4.4.2.4 send_instruction()

```
void send_instruction (
    uint8_t instr )
```

send an instruction to the lcd

Parameters

<i>instr</i>	the instruction
--------------	-----------------

4.4.2.5 set_address()

```
void set_address (
    uint8_t addr )
```

set the column in the current page

Parameters

<i>addr</i>	the column, from 0 to 63
-------------	--------------------------

4.4.2.6 set_page()

```
void set_page (
    uint8_t page )
```

select the page

Parameters

<i>page</i>	the numer of the page, from 0 to 7
-------------	------------------------------------

4.5 main.c File Reference

Main.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <util/delay_basic.h>
#include "wait.h"
#include "input.h"
#include "rendering.h"
#include "sprites.h"
#include "combat.h"
```

Macros

- `#define SHOOT_TIMEOUT 500`
- `#define MONSTER_COUNT 10`
- `#define WALL_COUNT 3`
- `#define MAX_WALL_LIFES 3`
- `#define DELAY_TIME 20`
- `#define BUTTON_CLICKED 1023`
- `#define SCREEN_WIDTH 128`
- `#define SCREEN_HEIGHT 64`
- `#define SCREEN_WIDTH_CENTER 64`
- `#define PAUSE_DURATION 800`
- `#define DEFAULT_BOUND_LEFT 0`
- `#define DEFAULT_BOUND_RIGHT 35`

Functions

- void [onCollide](#) (uint8_t index)
The Callback function which is executed if an object is hit by a laser ray.
- void [init_game](#) (void)
Reset all variables for a clean start of the game.
- void [drawScore](#) (uint8_t x, uint8_t y)
Draws the score on the screen.
- void [drawLifes](#) (void)
Draws the remaining lifes of the player as heart symbols on the screen.
- void [player_movement](#) (void)

- Handles the movement of the player.*
 - void `calculate_bounds` (void)
- Calculates how far the monsters can move, depending on which of them are alive.*
 - void `shoot_monsters` (void)
- Generates random shots by alive monsters.*
 - void `animatePlayer` (void)
- Makes the player blink.*
 - void `renderGame` (void)
- Renders the game.*
 - void `renderGameOver` (void)
- Renders the "Game Over" screen.*
 - int `main` (void)

Variables

- `collider default_monsters` [MONSTER_COUNT]
- `collider monsters` [MONSTER_COUNT]
- `bool dead_monsters` [MONSTER_COUNT]
- `uint8_t count_dead_monsters` = 0
- `uint8_t monster_colliders` [MONSTER_COUNT]
- `uint8_t monster_speed` = 3
- `int8_t monsters_left_bound` = DEFAULT_BOUND_LEFT
- `int8_t monsters_right_bound` = DEFAULT_BOUND_RIGHT
- `uint8_t monsters_movement_sleep` = 0
- `uint8_t monsters_movement_direction` = 1
- `collider default_walls` [WALL_COUNT]
- `collider walls` [WALL_COUNT]
- `uint8_t wall_colliders` [WALL_COUNT]
- `uint8_t wall_state` [WALL_COUNT] = {0,0,0}
- `int score` = 0
- `uint64_t gameTime` = 0
- `uint64_t gameTimePrev` = 0
- `uint64_t pauseUntil` = 0
- `bool pressed` = false
- `uint64_t shootTimeout` = 0
- `collider default_player` = {SCREEN_WIDTH_CENTER - 6, SCREEN_HEIGHT - 11, 13, 11}
- `collider player`
- `uint8_t playerCollider`
- `bool playerFlashing` = false
- `uint16_t playerFlashCount` = 0
- `uint8_t lives` = 3

4.5.1 Detailed Description

Main.

4.5.2 Function Documentation

4.5.2.1 animatePlayer()

```
void animatePlayer (
    void )
```

Makes the player blink.

4.5.2.2 calculate_bounds()

```
void calculate_bounds (
    void )
```

Calculates how far the monsters can move, depending on which of them are alive.

4.5.2.3 drawLives()

```
void drawLives (
    void )
```

Draws the remaining lives of the player as heart symbols on the screen.

4.5.2.4 drawScore()

```
void drawScore (
    uint8_t x,
    uint8_t y )
```

Draws the score on the screen.

Parameters

<i>x</i>	X-Coordinate on the screen.
<i>y</i>	Y-Coordinate on the screen.

4.5.2.5 init_game()

```
void init_game (
    void )
```

Reset all variables for a clean start of the game.

4.5.2.6 onCollide()

```
void onCollide (
    uint8_t index )
```

The Callback function which is executed if an object is hit by a laser ray.

This functions handles collisions between objects and laser rays. If the figure of the player is hit, the game is paused and one life is removed. If a monster is hit, it gets deleted. Depending on the amount of monsters, the speed is increased. If a wall is hit, the state which represents the amount of destruction is increased and the wall is deleted if necessary.

Parameters

<i>index</i>	The index of the collider which has been hit.
--------------	---

4.5.2.7 player_movement()

```
void player_movement (
    void )
```

Handles the movement of the player.

First, the x value of the Joystick is retrieved. This value is translated into a left/right movement or a shot. This function also ensures that the player cannot move out of the display

4.5.2.8 renderGame()

```
void renderGame (
    void )
```

Renders the game.

This function handles the game rendering and calls all necessary functions for the gameplay.

4.5.2.9 renderGameOver()

```
void renderGameOver (
    void )
```

Renders the "Game Over" screen.

Writes "Game Over" and the reached score on the screen. If the Joystick is clicked, the Game gets restarted.

4.5.2.10 shoot_monsters()

```
void shoot_monsters (
    void )
```

Generates random shots by alive monsters.

4.5.3 Variable Documentation

4.5.3.1 default_monsters

```
collider default_monsters[MONSTER_COUNT]
```

Initial value:

```
= {
    {0, 10, 13, 9},
    {20, 10, 13, 9},
    {40, 10, 13, 9},
    {60, 10, 13, 9},
    {80, 10, 13, 9},
    {0, 25, 13, 9},
    {20, 25, 13, 9},
    {40, 25, 13, 9},
    {60, 25, 13, 9},
    {80, 25, 13, 9}
}
```

4.5.3.2 default_walls

```
collider default_walls[WALL_COUNT]
```

Initial value:

```
= {
    {20, 45, 13, 5},
    {64 - 6, 45, 13, 5},
    {94, 45, 13, 5}
}
```

4.6 rendering.h File Reference

Display API.

```
#include <avr/io.h>
#include <stdbool.h>
```

Functions

- void `init_buffer` (void)
Initializes screen communication and screen buffer. Call once at the beginning.
- void `flash_buffer` (void)
Flashes the content of the buffer to the lcd.
- void `clear_buffer` (void)
clears the buffer, setting all values to zero
- void `set_pixel` (uint8_t x, uint8_t y, bool active)
Sets the value of a pixel in the screen buffer to either on or off, depending on the active param.
- void `fill_rect` (uint8_t x, uint8_t y, uint8_t width, uint8_t height, bool active)
Fills a rectangle, setting all pixels to either on or off, depending on the active param.

4.6.1 Detailed Description

Display API.

4.6.2 Function Documentation

4.6.2.1 fill_rect()

```
void fill_rect (
    uint8_t _x,
    uint8_t _y,
    uint8_t width,
    uint8_t height,
    bool active )
```

Fills a rectangle, setting all pixels to either on or off, depending on the active param.

Parameters

<i>_x</i>	the x position of the rect
<i>_y</i>	the y position of the rect
<i>width</i>	the width of the rect
<i>height</i>	the height of the rect
<i>active</i>	true if the pixel should be white, false otherwise

4.6.2.2 set_pixel()

```
void set_pixel (
    uint8_t x,
    uint8_t y,
    bool active )
```

Sets the value of a pixel in the screen buffer to either on or off, depending on the active param.

Parameters

<i>x</i>	the x position of the pixel
<i>y</i>	the y position of the pixel
<i>active</i>	true if the pixel should be white, false otherwise

4.7 sprites.h File Reference

Sprites for the game.

Classes

- struct [Sprite](#)
Representing a [Sprite](#) which can be drawn onto the display.

Macros

- #define [MAX_SPRITE_HEIGHT](#) 16
The maximum allowed height of a sprite.

Typedefs

- typedef struct [Sprite](#) [sprite](#)
Representing a [Sprite](#) which can be drawn onto the display.

Functions

- void [draw_sprite](#) (uint8_t x, uint8_t y, const [sprite](#) *figure)
draw the given sprite at the given position

Variables

- const [sprite](#) [MONSTER](#)
The sprite of a monster.
- const [sprite](#) [MONSTER2](#)
The sprite of a monster.
- const [sprite](#) [PLAYER](#)
The sprite of the players figure.
- const [sprite](#) [NUMBERS](#) []
- const [sprite](#) [WALL](#)
The sprite of a wall without any damages.
- const [sprite](#) [WALL_DAMAGE_1](#)
The sprite of a wall with low damages.
- const [sprite](#) [WALL_DAMAGE_2](#)
The sprite of a wall with a higher amount of damages.
- const [sprite](#) [WALL_DAMAGE_3](#)
The sprite of a wall with the highest amount of possible damages.
- const [sprite](#) [HEART](#)
The sprite of a heart.
- const [sprite](#) [LETTER_G](#)
The sprite for the letter G.
- const [sprite](#) [LETTER_A](#)
The sprite for the letter A.

- const [sprite](#) `LETTER_M`
The sprite for the letter M.
- const [sprite](#) `LETTER_E`
The sprite for the letter E.
- const [sprite](#) `LETTER_O`
The sprite for the letter O.
- const [sprite](#) `LETTER_V`
The sprite for the letter V.
- const [sprite](#) `LETTER_R`
The sprite for the letter R.

4.7.1 Detailed Description

Sprites for the game.

4.7.2 Typedef Documentation

4.7.2.1 `sprite`

```
typedef struct Sprite sprite
```

Representing a [Sprite](#) which can be drawn onto the display.

Parameters

<i>data</i>	An Array which contains uint16_t variables, matching the pixels of a row
<i>width</i>	The width of a sprite in pixels
<i>height</i>	The height of a sprite in pixels

4.7.3 Function Documentation

4.7.3.1 `draw_sprite()`

```
void draw_sprite (
    uint8_t x,
    uint8_t y,
    const sprite * figure )
```

draw the given sprite at the given position

Parameters

<i>x</i>	the x position
<i>y</i>	the y position
<i>figure</i>	a pointer to the sprite

draw the given sprite at the given position

Parameters

<i>x</i>	Coordinate
<i>y</i>	Coordinate
<i>figure</i>	The figure that should be drawn

4.7.4 Variable Documentation

4.7.4.1 HEART

```
const sprite HEART
```

The sprite of a heart.

4.7.4.2 LETTER_A

```
const sprite LETTER_A
```

The sprite for the letter A.

4.7.4.3 LETTER_E

```
const sprite LETTER_E
```

The sprite for the letter E.

4.7.4.4 LETTER_G

```
const sprite LETTER_G
```

The sprite for the letter G.

4.7.4.5 LETTER_M

```
const sprite LETTER_M
```

The sprite for the letter M.

4.7.4.6 LETTER_O

```
const sprite LETTER_O
```

The sprite for the letter O.

4.7.4.7 LETTER_R

```
const sprite LETTER_R
```

The sprite for the letter R.

4.7.4.8 LETTER_V

```
const sprite LETTER_V
```

The sprite for the letter V.

4.7.4.9 MONSTER

```
const sprite MONSTER
```

The sprite of a monster.

4.7.4.10 MONSTER2

```
const sprite MONSTER2
```

The sprite of a monster.

4.7.4.11 PLAYER

```
const sprite PLAYER
```

The sprite of the players figure.

4.7.4.12 WALL

```
const sprite WALL
```

The sprite of a wall without any damages.

4.7.4.13 WALL_DAMAGE_1

```
const sprite WALL_DAMAGE_1
```

The sprite of a wall with low damages.

4.7.4.14 WALL_DAMAGE_2

```
const sprite WALL_DAMAGE_2
```

The sprite of a wall with a higher amount of damages.

4.7.4.15 WALL_DAMAGE_3

```
const sprite WALL_DAMAGE_3
```

The sprite of a wall with the highest amount of possible damages.

4.8 wait.h File Reference

Delay.

Functions

- void `delay_millis` (uint16_t ms)
Wait for the given amount of milliseconds.

4.8.1 Detailed Description

Delay.

4.8.2 Function Documentation

4.8.2.1 `delay_millis()`

```
void delay_millis (
    uint16_t ms )
```

Wait for the given amount of milliseconds.

Parameters

<i>ms</i>	the amount of milliseconds to wait
-----------	------------------------------------

Index

animate_lasers
 combat.h, 8
animatePlayer
 main.c, 16
asserts.h, 7

calculate_bounds
 main.c, 17
clear_page
 lcd.h, 13
collider, 5
combat.h, 7
 animate_lasers, 8
 register_collide_callback, 8
 register_collider, 9
 remove_collider, 9
 reset_combat, 9
 shoot_laser, 9

default_monsters
 main.c, 19
default_walls
 main.c, 19
delay_millis
 wait.h, 26
display_on
 lcd.h, 13
draw_sprite
 sprites.h, 22
drawLives
 main.c, 17
drawScore
 main.c, 17

fill_rect
 rendering.h, 20

get_input
 input.h, 10

HEART
 sprites.h, 23

init_game
 main.c, 17
init_input
 input.h, 12
input.h, 10
 get_input, 10
 init_input, 12

laser, 5
lcd.h, 12
 clear_page, 13
 display_on, 13
 send_data, 13
 send_instruction, 14
 set_address, 14
 set_page, 14
LETTER_A
 sprites.h, 23
LETTER_E
 sprites.h, 23
LETTER_G
 sprites.h, 23
LETTER_M
 sprites.h, 23
LETTER_O
 sprites.h, 24
LETTER_R
 sprites.h, 24
LETTER_V
 sprites.h, 24

main.c, 15
 animatePlayer, 16
 calculate_bounds, 17
 default_monsters, 19
 default_walls, 19
 drawLives, 17
 drawScore, 17
 init_game, 17
 onCollide, 17
 player_movement, 18
 renderGame, 18
 renderGameOver, 18
 shoot_monsters, 18
MONSTER
 sprites.h, 24
MONSTER2
 sprites.h, 24

onCollide
 main.c, 17

PLAYER
 sprites.h, 24
player_movement
 main.c, 18

register_collide_callback

- combat.h, [8](#)
- register_collider
 - combat.h, [9](#)
- remove_collider
 - combat.h, [9](#)
- renderGame
 - main.c, [18](#)
- renderGameOver
 - main.c, [18](#)
- rendering.h, [19](#)
 - fill_rect, [20](#)
 - set_pixel, [20](#)
- reset_combat
 - combat.h, [9](#)
- send_data
 - lcd.h, [13](#)
- send_instruction
 - lcd.h, [14](#)
- set_address
 - lcd.h, [14](#)
- set_page
 - lcd.h, [14](#)
- set_pixel
 - rendering.h, [20](#)
- shoot_laser
 - combat.h, [9](#)
- shoot_monsters
 - main.c, [18](#)
- Sprite, [6](#)
- sprite
 - sprites.h, [22](#)
- sprites.h, [21](#)
 - draw_sprite, [22](#)
 - HEART, [23](#)
 - LETTER_A, [23](#)
 - LETTER_E, [23](#)
 - LETTER_G, [23](#)
 - LETTER_M, [23](#)
 - LETTER_O, [24](#)
 - LETTER_R, [24](#)
 - LETTER_V, [24](#)
 - MONSTER, [24](#)
 - MONSTER2, [24](#)
 - PLAYER, [24](#)
 - sprite, [22](#)
 - WALL, [25](#)
 - WALL_DAMAGE_1, [25](#)
 - WALL_DAMAGE_2, [25](#)
 - WALL_DAMAGE_3, [25](#)
- wait.h, [25](#)
 - delay_millis, [26](#)
- WALL
 - sprites.h, [25](#)
- WALL_DAMAGE_1
 - sprites.h, [25](#)
- WALL_DAMAGE_2
 - sprites.h, [25](#)
- WALL_DAMAGE_3
 - sprites.h, [25](#)

- WALL_DAMAGE_3
 - sprites.h, [25](#)