HoldraszálláSCH

NHF3

Generated by Doxygen 1.9.5

1 HoldraszálláSCH 1

1.1 Fordítás . . . 1

1.1.1 Make . . . 1

1.1.2 Manuálisan . . . 1

1.2 Játékmenet . . . 1

1.2.1 Kezdő állapot . . . 1

1.2.2 Leszállás . . . 2

1.2.3 A játék vége . . . 2

1.2.4 Irányítás . . . 2

1.2.5 Pontozás . . . 2

2 Data Structure Index 3

2.1 Data Structures . . . 3

3 File Index 5

3.1 File List . . . 5

4 Data Structure Documentation 7

4.1 Button Struct Reference . . . 7

4.1.1 Detailed Description . . . 7

4.1.2 Field Documentation . . . 7

4.1.2.1 hover . . . 7

4.1.2.2 rect . . . 7

4.1.2.3 text . . . 8

4.2 Camera Struct Reference . . . 8

4.2.1 Detailed Description . . . 8

4.2.2 Field Documentation . . . 8

4.2.2.1 height . . . 8

4.2.2.2 position . . . 8

4.2.2.3 renderer . . . 9

4.2.2.4 width . . . 9

4.2.2.5 zoom . . . 9

4.3 GameState Struct Reference . . . 9

4.3.1 Field Documentation . . . 9

4.3.1.1 camera . . . 9

4.3.1.2 delta_time . . . 10

4.3.1.3 destroyed . . . 10

4.3.1.4 game_over . . . 10

4.3.1.5 game_over_dealy . . . 10

4.3.1.6 lander . . . 10

4.3.1.7 saved . . . 10

4.3.1.8 successfull . . . 10

4.3.1.9 time_started . . . 11

4.4 ImpactPoint Struct Reference 11
4.4.1 Field Documentation 11
4.4.1.1 can_collide 11
4.4.1.2 point 11
4.5 Lander Struct Reference 11
4.5.1 Field Documentation 12
4.5.1.1 angular_velocity 12
4.5.1.2 dry_mass 12
4.5.1.3 engines 12
4.5.1.4 impact_count 12
4.5.1.5 particle_system 12
4.5.1.6 position 13
4.5.1.7 propellant 13
4.5.1.8 rotation 13
4.5.1.9 velocity 13
4.6 List Struct Reference 13
4.6.1 Field Documentation 13
4.6.1.1 first 13
4.6.1.2 last 14
4.7 ListElement Struct Reference 14
4.7.1 Field Documentation 14
4.7.1.1 next 14
4.7.1.2 particle 14
4.7.1.3 prev 14
4.8 Particle Struct Reference 15
4.8.1 Field Documentation 15
4.8.1.1 end_color 15
4.8.1.2 life_time 15
4.8.1.3 lived 15
4.8.1.4 position 15
4.8.1.5 size 15
4.8.1.6 start_color 16
4.8.1.7 velocity 16
4.9 Score Struct Reference 16
4.9.1 Detailed Description 16
4.9.2 Field Documentation 16
4.9.2.1 fuel 16
4.9.2.2 name 17
4.9.2.3 quality 17
4.9.2.4 score 17
4.9.2.5 time 17
4.10 Vector2 Struct Reference 17

4.10.1 Field Documentation . . . 17
4.10.1.1 x . . . 17
4.10.1.2 y . . . 17

5 File Documentation 19

5.1 include/button.h File Reference . . . 19
5.1.1 Typedef Documentation . . . 19
5.1.1.1 Button . . . 19
5.1.2 Function Documentation . . . 20
5.1.2.1 render_button() . . . 20
5.2 button.h . . . 20
5.3 include/camera.h File Reference . . . 20
5.3.1 Typedef Documentation . . . 21
5.3.1.1 Camera . . . 21
5.3.2 Function Documentation . . . 21
5.3.2.1 get_screen_coordinates() . . . 21
5.3.2.2 get_world_coordinates() . . . 22
5.3.2.3 lerp() . . . 22
5.3.2.4 update_camera() . . . 22
5.3.3 Variable Documentation . . . 23
5.3.3.1 PIXELS_PER_METER . . . 23
5.4 camera.h . . . 23
5.5 include/events.h File Reference . . . 24
5.5.1 Typedef Documentation . . . 24
5.5.1.1 EventCode . . . 24
5.5.2 Enumeration Type Documentation . . . 24
5.5.2.1 EventCode . . . 24
5.6 events.h . . . 24
5.7 include/file_handler.h File Reference . . . 25
5.7.1 Typedef Documentation . . . 25
5.7.1.1 Score . . . 25
5.7.2 Function Documentation . . . 25
5.7.2.1 append_score() . . . 25
5.7.2.2 read_scores() . . . 26
5.8 file_handler.h . . . 26
5.9 include/game.h File Reference . . . 26
5.9.1 Typedef Documentation . . . 27
5.9.1.1 GameState . . . 27
5.9.2 Function Documentation . . . 27
5.9.2.1 calculate_score() . . . 27
5.9.2.2 destroy_game() . . . 27
5.9.2.3 game_events() . . . 27

5.9.2.4 init_game() 27
5.9.2.5 landing_quality() 28
5.9.2.6 render_game_over() 28
5.9.2.7 save_state() 28
5.9.2.8 update_game() 28
5.10 game.h 28
5.11 include/lander.h File Reference 29
5.11.1 Typedef Documentation 30
5.11.1.1 ImpactPoint 30
5.11.1.2 Lander 30
5.11.2 Enumeration Type Documentation 30
5.11.2.1 Engine 30
5.11.3 Function Documentation 30
5.11.3.1 bulk_add_particles() 30
5.11.3.2 destroy_lander() 31
5.11.3.3 display_dashboard() 31
5.11.3.4 get_impact_force() 31
5.11.3.5 get_lander_inertia() 32
5.11.3.6 get_torque() 32
5.11.3.7 init_lander() 32
5.11.3.8 lander_to_world_coord() 33
5.11.3.9 lander_total_mass() 33
5.11.3.10 render_lander() 33
5.11.3.11 update_lander() 34
5.12 lander.h 34
5.13 include/leaderboard.h File Reference 35
5.13.1 Function Documentation 35
5.13.1.1 destroy_leaderboard() 35
5.13.1.2 init_leaderboard() 35
5.13.1.3 render_leaderboard() 36
5.14 leaderboard.h 36
5.15 include/main.h File Reference 36
5.15.1 Typedef Documentation 36
5.15.1.1 Screen 36
5.15.2 Enumeration Type Documentation 36
5.15.2.1 Screen 36
5.16 main.h 37
5.17 include/menu.h File Reference 37
5.17.1 Typedef Documentation 37
5.17.1.1 Screen 38
5.17.2 Enumeration Type Documentation 38
5.17.2.1 Screen 38

5.17.3 Function Documentation 38
5.17.3.1 destroy_menu() 38
5.17.3.2 init_menu() 38
5.17.3.3 menu_events() 38
5.17.3.4 render_menu() 39
5.18 menu.h 39
5.19 include/particle.h File Reference 39
5.19.1 Typedef Documentation 40
5.19.1.1 List 40
5.19.1.2 ListElement 40
5.19.1.3 Particle 40
5.19.2 Function Documentation 40
5.19.2.1 append_particle() 40
5.19.2.2 delete_particle() 40
5.19.2.3 destroy_particles() 40
5.19.2.4 render_particles() 41
5.19.2.5 update_particles() 41
5.20 particle.h 41
5.21 include/terrain.h File Reference 41
5.21.1 Function Documentation 42
5.21.1.1 get_terrain_height() 42
5.21.1.2 init_terrain() 42
5.21.1.3 render_terrain() 42
5.21.2 Variable Documentation 43
5.21.2.1 TERRAIN_SEED 43
5.22 terrain.h 43
5.23 include/text_io.h File Reference 43
5.23.1 Function Documentation 43
5.23.1.1 input_text() 44
5.23.1.2 render_text_centered() 44
5.24 text_io.h 44
5.25 include/vector.h File Reference 44
5.25.1 Typedef Documentation 45
5.25.1.1 Vector2 45
5.25.2 Function Documentation 45
5.25.2.1 V_add() 45
5.25.2.2 V_cross_product() 46
5.25.2.3 V_divide_const() 46
5.25.2.4 V_len() 46
5.25.2.5 V_multiply() 46
5.25.2.6 V_multiply_const() 46
5.25.2.7 V_normalize() 47

5.25.2.8 V_rotate() 47
5.25.2.9 V_subtract() 47
5.25.2.10 V_to_point() 47
5.26 vector.h 48
5.27 README.md File Reference 48
5.28 src/button.c File Reference 48
5.28.1 Function Documentation 49
5.28.1.1 render_button() 49
5.28.2 Variable Documentation 49
5.28.2.1 bg_color 49
5.28.2.2 fg_color 49
5.28.2.3 hover_color 49
5.29 src/camera.c File Reference 49
5.29.1 Function Documentation 50
5.29.1.1 get_screen_coordinates() 50
5.29.1.2 get_world_coordinates() 50
5.29.1.3 lerp() 51
5.29.1.4 lerp2() 51
5.29.1.5 update_camera() 51
5.29.2 Variable Documentation 51
5.29.2.1 camera_speed 52
5.29.2.2 PIXELS_PER_METER 52
5.30 src/file_handler.c File Reference 52
5.30.1 Function Documentation 52
5.30.1.1 append_score() 52
5.30.1.2 read_scores() 52
5.31 src/game.c File Reference 53
5.31.1 Function Documentation 53
5.31.1.1 calculate_score() 53
5.31.1.2 destroy_game() 54
5.31.1.3 game_events() 54
5.31.1.4 init_game() 54
5.31.1.5 landing_quality() 54
5.31.1.6 render_game_over() 54
5.31.1.7 save_state() 55
5.31.1.8 update_game() 55
5.31.2 Variable Documentation 55
5.31.2.1 font_large 55
5.31.2.2 font_small 55
5.32 src/lander.c File Reference 55
5.32.1 Function Documentation 56
5.32.1.1 bulk_add_particles() 57

5.32.1.2 destroy_lander() 57
5.32.1.3 display_dashboard() 57
5.32.1.4 get_impact_force() 57
5.32.1.5 get_lander_inertia() 58
5.32.1.6 get_torque() 58
5.32.1.7 init_lander() 58
5.32.1.8 lander_to_world_coord() 59
5.32.1.9 lander_total_mass() 59
5.32.1.10 render_lander() 59
5.32.1.11 to_metric() 60
5.32.1.12 update_lander() 60
5.32.2 Variable Documentation 60
5.32.2.1 center_of_mass 60
5.32.2.2 dashboard_texture 60
5.32.2.3 dry_mass 60
5.32.2.4 friction_coefficient 61
5.32.2.5 g 61
5.32.2.6 inertia_max 61
5.32.2.7 inertia_min 61
5.32.2.8 lander_texture 61
5.32.2.9 main_engine_fuel_rate 61
5.32.2.10 main_engine_thrust 61
5.32.2.11 propellant_mass 61
5.32.2.12 rcs_fuel_rate 62
5.32.2.13 rcs_thrust 62
5.32.2.14 size_px 62
5.33 src/leaderboard.c File Reference 62
5.33.1 Function Documentation 62
5.33.1.1 destroy_leaderboard() 62
5.33.1.2 init_leaderboard() 63
5.33.1.3 render_leaderboard() 63
5.33.2 Variable Documentation 63
5.33.2.1 count 63
5.33.2.2 font 63
5.33.2.3 score 63
5.34 src/main.c File Reference 63
5.34.1 Function Documentation 64
5.34.1.1 main() 64
5.35 src/menu.c File Reference 64
5.35.1 Function Documentation 64
5.35.1.1 destroy_menu() 64
5.35.1.2 init_menu() 64

5.35.1.3 menu_events() 65
5.35.1.4 render_menu() 65
5.35.2 Variable Documentation 65
5.35.2.1 button_count 65
5.35.2.2 buttons 65
5.35.2.3 font 65
5.35.2.4 margin 65
5.36 src/particle.c File Reference 66
5.36.1 Function Documentation 66
5.36.1.1 append_particle() 66
5.36.1.2 delete_particle() 66
5.36.1.3 destroy_particles() 66
5.36.1.4 render_particles() 67
5.36.1.5 update_particles() 67
5.37 src/terrain.c File Reference 67
5.37.1 Function Documentation 67
5.37.1.1 get_terrain_height() 67
5.37.1.2 init_terrain() 68
5.37.1.3 noise() 68
5.37.1.4 pseudo_random() 68
5.37.1.5 render_terrain() 68
5.37.2 Variable Documentation 68
5.37.2.1 terrain_max_height 69
5.37.2.2 TERRAIN_SEED 69
5.38 src/text_io.c File Reference 69
5.38.1 Function Documentation 69
5.38.1.1 input_text() 69
5.38.1.2 render_text_centered() 70
5.38.1.3 valid() 70
5.39 src/vector.c File Reference 70
5.39.1 Function Documentation 71
5.39.1.1 V_add() 71
5.39.1.2 V_cross_product() 71
5.39.1.3 V_divide_const() 71
5.39.1.4 V_len() 71
5.39.1.5 V_multiply() 71
5.39.1.6 V_multiply_const() 72
5.39.1.7 V_normalize() 72
5.39.1.8 V_rotate() 72
5.39.1.9 V_subtract() 72
5.39.1.10 V_to_point() 72

Index **73**

Chapter 1

HoldraszálláSCH

1.1 Fordítás

1.1.1 Make

A projekthez tartozik egy Makefile, ennek segítségével viszonylag egyszerű lefordítani a projektet.

Linux

Ubuntu-n lett tesztelve a makefile, de elméletileg más disztibúción is működik. Elég belépnünk a projekt mappájába és kiadni a `make` parancsot. Az elkészült futtatható fájl és az egyéb futáshoz szükséges fájlok a `build/bin/` mappában lesznek megtalálhatóak.

Windows

Windows-on egy kicsit bonyolultabb, itt először a `MINGW_LIB` és az `SDL_INCLUDE` környezeti változókban meg kell adnunk a megfelelő elérési utat.

Powershell-ben ezt a következő parancsok segítségével tehetjük meg:

```
$env:MINGW_LIB = "C:\Program Files\CodeBlocks\MinGW\lib"
$env:SDL_INCLUDE = "C:\Program Files\CodeBlocks\MinGW\include\SDL2"
```

Ezután be kell lépnünk a projekt mappájába majd a `mingw32-make` parancs segítségével fordíthatjuk a projektet. Az elkészült `.exe` fájl és a futáshoz szükséges egyéb fájlok a `build\bin\` mappában lesznek megtalálhatóak.

1.1.2 Manuálisan

A forrás fájlok az `src`, a header fájlok pedig az `include` mappában találhatóak. Ezen kívül a fordításhoz szükség van még az SDL2 grafikus könyvtárra. Futtatáshoz pedig fontos, hogy az elkészült futtatható fájl és az `assets` mappa egy mappában legyenek.

1.2 Játékmenet

1.2.1 Kezdő állapot

A Hold felszínét oldalnézetben rajzoljuk ki. A leszállóegység a képernyő bal felső sarkában van, nagy sebességgel halad jobbra és kezdetben nincs lefele irányú sebessége.

1.2.2 Leszállás

A leszállóegységre állandó lefele irányuló gyorsulási erő hat. Feladatunk a kezdeti vízszintes irányú sebesség csökkentése, illetve az ereszkedés sebességének szabályozása, mindezt úgy, hogy kellően egyenletes terepen tudjunk landolni. Ha elég közel kerültünk a felszínhez a játék közelebbi nézetre vált, ezzel segítve a pontosabb manőverezést.

A leszállóegység sebességét a fő hajtómű segítségével szabályozhatjuk. Kisebb manővereket, illetve a leszállóegység forgatását az oldalán található kis hajtóművek segítségével végezhetjük. A hajtóművek által kifejtett erő állandó, azonban ahogy fogy az üzemanyag a leszállóegység egyre könnyebb lesz, így a rá ható erő nagyobb mértékben befolyásolja a gyorsulást.

1.2.3 A játék vége

A játék mindenképpen véget ér amikor a leszállóegység eléri a felszínt. A leszállás sikeresnek minősül, ha a leszállóegység vízszintes talajra érkezik, és elég kicsi a vízszintes és függőleges sebessége is. A leszállóegység megsemmisül, vagyis a játékos veszít, ha túl nagy sebességgel csapódik be, vagy nem egyenletes talajra próbál leszállni, vagy túl nagy az elfordulása.

1.2.4 Irányítás

A hajtóműveket billentyűk lenyomva tartásával irányíthatjuk.

- Fő hajtómű: `W`
- Jobb oldali hajtómű: `A`
- Bal oldali hajtómű: `D`
- Jobbra fordulás: `K`
- Balra fordulás: `J`

A hajtóművek egy, a hajtómű irányával ellentétes irányú erőt fejtenek ki a leszállóegységre.

1.2.5 Pontozás

A játékos által szerzett pontokat a leszállás minőségéből (mekkora volt a sebessége, milyen volt az elfordulása) illetve a felhasznált üzemanyag mennyiségéből számítjuk ki.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Button
Structure for creating a button 7
Camera
The structure containing the camera 8
GameState 9
ImpactPoint 11
Lander 11
List 13
ListElement 14
Particle 15
Score
Structure to help save and load scoreboard entries 16
Vector2 17

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/button.h 19
include/camera.h 20
include/events.h 24
include/file_handler.h 25
include/game.h 26
include/lander.h 29
include/leaderboard.h 35
include/main.h 36
include/menu.h 37
include/particle.h 39
include/terrain.h 41
include/text_io.h 43
include/vector.h 44
src/button.c 48
src/camera.c 49
src/file_handler.c 52
src/game.c 53
src/lander.c 55
src/leaderboard.c 62
src/main.c 63
src/menu.c 64
src/particle.c 66
src/terrain.c 67
src/text_io.c 69
src/vector.c 70

Chapter 4

Data Structure Documentation

4.1 Button Struct Reference

Structure for creating a button.

```
#include <button.h>
```

Data Fields

- SDL_Rect rect
- bool hover
- char ∗ text

4.1.1 Detailed Description

Structure for creating a button.

4.1.2 Field Documentation

4.1.2.1 hover

```
bool hover
```

4.1.2.2 rect

```
SDL_Rect rect
```

4.1.2.3 text

```
char* text
```

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

- include/button.h

4.2 Camera Struct Reference

The structure containing the camera.

```
#include <camera.h>
```

Data Fields

- Vector2 position
- double zoom
- SDL_Renderer * renderer
- double width
- double height

4.2.1 Detailed Description

The structure containing the camera.

4.2.2 Field Documentation

4.2.2.1 height

```
double height
```

4.2.2.2 position

```
Vector2 position
```

4.2.2.3 renderer

```
SDL_Renderer* renderer
```

4.2.2.4 width

```
double width
```

4.2.2.5 zoom

```
double zoom
```

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

- include/camera.h

4.3 GameState Struct Reference

```
#include <game.h>
```

Data Fields

- Lander lander
- Camera camera
- double time_started
- double delta_time
- double game_over_dealy
- bool game_over
- bool successfull
- bool destroyed
- bool saved

4.3.1 Field Documentation

4.3.1.1 camera

```
Camera camera
```

4.3.1.2 delta_time

```
double delta_time
```

4.3.1.3 destroyed

```
bool destroyed
```

4.3.1.4 game_over

```
bool game_over
```

4.3.1.5 game_over_dealy

```
double game_over_dealy
```

4.3.1.6 lander

```
Lander lander
```

4.3.1.7 saved

```
bool saved
```

4.3.1.8 successfull

```
bool successfull
```

4.3.1.9 time_started

```
double time_started
```

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

- include/game.h

4.4 ImpactPoint Struct Reference

```
#include <lander.h>
```

Data Fields

- Vector2 point
- bool can_collide

4.4.1 Field Documentation

4.4.1.1 can_collide

```
bool can_collide
```

4.4.1.2 point

```
Vector2 point
```

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

- include/lander.h

4.5 Lander Struct Reference

```
#include <lander.h>
```

Data Fields

- Vector2 position
- Vector2 velocity
- double rotation
- double angular_velocity
- double dry_mass
- double propellant
- int impact_count
- bool engines [5]
- List particle_system

4.5.1 Field Documentation

4.5.1.1 angular_velocity

```
double angular_velocity
```

4.5.1.2 dry_mass

```
double dry_mass
```

4.5.1.3 engines

```
bool engines[5]
```

4.5.1.4 impact_count

```
int impact_count
```

4.5.1.5 particle_system

```
List particle_system
```

4.5.1.6 position

```
Vector2 position
```

4.5.1.7 propellant

```
double propellant
```

4.5.1.8 rotation

```
double rotation
```

4.5.1.9 velocity

```
Vector2 velocity
```

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

- include/lander.h

4.6 List Struct Reference

```
#include <particle.h>
```

Data Fields

- ListElement ∗ first
- ListElement ∗ last

4.6.1 Field Documentation

4.6.1.1 first

```
ListElement∗ first
```

4.6.1.2 last

```
ListElement* last
```

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

- include/particle.h

4.7 ListElement Struct Reference

```
#include <particle.h>
```

Data Fields

- Particle particle
- ListElement ∗ next
- ListElement ∗ prev

4.7.1 Field Documentation

4.7.1.1 next

```
ListElement* next
```

4.7.1.2 particle

```
Particle particle
```

4.7.1.3 prev

```
ListElement* prev
```

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

- include/particle.h

4.8 Particle Struct Reference

```
#include <particle.h>
```

Data Fields

- SDL_Color start_color
- SDL_Color end_color
- Vector2 velocity
- Vector2 position
- double life_time
- double lived
- double size

4.8.1 Field Documentation

4.8.1.1 end_color

```
SDL_Color end_color
```

4.8.1.2 life_time

```
double life_time
```

4.8.1.3 lived

```
double lived
```

4.8.1.4 position

```
Vector2 position
```

4.8.1.5 size

```
double size
```

4.8.1.6 start_color

```
SDL_Color start_color
```

4.8.1.7 velocity

```
Vector2 velocity
```

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

- include/particle.h

4.9 Score Struct Reference

Structure to help save and load scoreboard entries.

```
#include <file_handler.h>
```

Data Fields

- int score
- char name [15]
- double time
- double fuel
- double quality

4.9.1 Detailed Description

Structure to help save and load scoreboard entries.

4.9.2 Field Documentation

4.9.2.1 fuel

```
double fuel
```

4.9.2.2 name

```
char name[15]
```

4.9.2.3 quality

```
double quality
```

4.9.2.4 score

```
int score
```

4.9.2.5 time

```
double time
```

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

- include/file_handler.h

4.10 Vector2 Struct Reference

```
#include <vector.h>
```

Data Fields

- double x
- double y

4.10.1 Field Documentation

4.10.1.1 x

```
double x
```

4.10.1.2 y

```
double y
```

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

- include/vector.h

Chapter 5

File Documentation

5.1 include/button.h File Reference

```
#include <SDL.h>
#include <SDL_ttf.h>
#include <stdbool.h>
```

Data Structures

- struct Button

  Structure for creating a button.

Typedefs

- typedef struct Button Button

  Structure for creating a button.

Functions

- void render_button (SDL_Renderer ∗renderer, TTF_Font ∗font, Button ∗button)

5.1.1 Typedef Documentation

5.1.1.1 Button

```
typedef struct Button Button
```

Structure for creating a button.

5.1.2 Function Documentation

5.1.2.1 render_button()

```
void render_button (
            SDL_Renderer * renderer,
            TTF_Font * font,
            Button * button )
```

5.2 button.h

Go to the documentation of this file.

```
#ifndef BUTTON_H
#define BUTTON_H

#include <SDL.h>
#include <SDL_ttf.h>
#include <stdbool.h>

/// @brief Structure for creating a button
typedef struct Button {
    SDL_Rect rect;
    bool hover;
    char *text;
} Button;

void render_button(SDL_Renderer *renderer, TTF_Font *font, Button *button);

#endif
```

5.3 include/camera.h File Reference

```
#include <SDL.h>
#include "vector.h"
```

Data Structures

- struct Camera

  The structure containing the camera.

Typedefs

- typedef struct Camera Camera

  The structure containing the camera.

Functions

- void update_camera (Camera *camera, Vector2 lander_pos, double dt)

  Updates the position and zoom level of the camera.
- Vector2 get_screen_coordinates (Camera *camera, Vector2 world_coordinates)

  Convert world coordinates to screen coordinates.
- Vector2 get_world_coordinates (Camera *camera, Vector2 screen_coordinates)

  Convert screen coordinates to world coordinates.
- double lerp (double a, double b, double t)

  Function to interpolate between two points.

Variables

- const double PIXELS_PER_METER

  Used for converting between screen space and world space.

5.3.1 Typedef Documentation

5.3.1.1 Camera

```
typedef struct Camera Camera
```

The structure containing the camera.

5.3.2 Function Documentation

5.3.2.1 get_screen_coordinates()

```
Vector2 get_screen_coordinates (
            Camera * camera,
            Vector2 world_coordinates )
```

Convert world coordinates to screen coordinates.

Parameters

camera	The camera struct used for calculations
world_coordinates	The point to be converted

Returns

The coordinates of the point in screen space

5.3.2.2 get_world_coordinates()

```
Vector2 get_world_coordinates (
            Camera * camera,
            Vector2 screen_coordinates )
```

Convert screen coordinates to world coordinates.

Parameters

camera	The camera struct used for calculations
screen_coordinates	the point to bo converted

Returns

The coordinates of the point in world space

5.3.2.3 lerp()

```
double lerp (
            double a,
            double b,
            double t )
```

Function to interpolate between two points.

Parameters

a	point a
b	point b
t	time (0-1)

Returns

a if t = 0 b if t = 1

5.3.2.4 update_camera()

```
void update_camera (
            Camera * camera,
            Vector2 lander_pos,
            double dt )
```

Updates the position and zoom level of the camera.

Parameters

camera	The camera struct to update
lander_pos	Positoin of the lander for tracking
dt	Time in seconds since the last frame

5.3.3 Variable Documentation

5.3.3.1 PIXELS_PER_METER

```
const double PIXELS_PER_METER  [extern]
```

Used for converting between screen space and world space.

5.4 camera.h

Go to the documentation of this file.

```
#ifndef CAMERA_H
#define CAMERA_H

#include <SDL.h>

#include "vector.h"

/// @brief The structure containing the camera
typedef struct Camera {
    Vector2 position;
    double zoom;
    SDL_Renderer *renderer;
    double width;
    double height;
} Camera;

/// @brief Used for converting between screen space and world space.
extern const double PIXELS_PER_METER;

/// @brief Updates the position and zoom level of the camera.
/// @param camera The camera struct to update
/// @param lander_pos Positoin of the lander for tracking
/// @param dt Time in seconds since the last frame
void update_camera(Camera *camera, Vector2 lander_pos, double dt);

/// @brief Convert world coordinates to screen coordinates.
/// @param camera The camera struct used for calculations
/// @param world_coordinates The point to be converted
/// @return The coordinates of the point in screen space
Vector2 get_screen_coordinates(Camera *camera, Vector2 world_coordinates);

/// @brief Convert screen coordinates to world coordinates.
/// @param camera The camera struct used for calculations
/// @param screen_coordinates the point to bo converted
/// @return The coordinates of the point in world space
Vector2 get_world_coordinates(Camera *camera, Vector2 screen_coordinates);

/// @brief Function to interpolate between two points
/// @param a point a
/// @param b point b
/// @param t time (0-1)
/// @return a if t = 0 b if t = 1
double lerp(double a, double b, double t);

#endif
```

5.5 include/events.h File Reference

Typedefs

- typedef enum EventCode EventCode

  Eventcodes used in game.

Enumerations

- enum EventCode { DEATH_EVENT_CODE , SUCCESS_EVENT_CODE }

  Eventcodes used in game.

5.5.1 Typedef Documentation

5.5.1.1 EventCode

```
typedef enum EventCode EventCode
```

Eventcodes used in game.

5.5.2 Enumeration Type Documentation

5.5.2.1 EventCode

```
enum EventCode
```

Eventcodes used in game.

Enumerator

DEATH_EVENT_CODE	
SUCCESS_EVENT_CODE	

5.6 events.h

Go to the documentation of this file.

```
#ifndef EVENTS_H
#define EVENTS_H

/// @brief Eventcodes used in game
```

```
typedef enum EventCode {
    DEATH_EVENT_CODE,
    SUCCESS_EVENT_CODE
} EventCode;

#endif
```

5.7 include/file_handler.h File Reference

`#include <stdio.h>`

Data Structures

- struct Score

  Structure to help save and load scoreboard entries.

Typedefs

- typedef struct Score Score

  Structure to help save and load scoreboard entries.

Functions

- void append_score (Score ∗score)
- int read_scores (Score ∗∗scores)

5.7.1 Typedef Documentation

5.7.1.1 Score

`typedef struct Score Score`

Structure to help save and load scoreboard entries.

5.7.2 Function Documentation

5.7.2.1 append_score()

```
void append_score (
            Score ∗ score )
```

5.7.2.2 read_scores()

```
int read_scores (
            Score ** scores )
```

5.8 file_handler.h

Go to the documentation of this file.

```
#ifndef FILEHANDLER_H
#define FILEHANDLER_H

#include <stdio.h>

/// @brief Structure to help save and load scoreboard entries
typedef struct Score {
    int score;
    char name[15];
     double time;
     double fuel;
     double quality;
} Score;

void append_score(Score *score);
int read_scores(Score **scores);
#endif
```

5.9 include/game.h File Reference

```
#include <SDL.h>
#include "lander.h"
#include "camera.h"
#include "menu.h"
```

Data Structures

- struct GameState

Typedefs

- typedef struct GameState GameState

Functions

- GameState init_game (SDL_Renderer ∗renderer, int ∗terrain_seed)

  Sets up the default parameters for the lander, the camera and the world.
- void update_game (GameState ∗state)
- Screen game_events (SDL_Event event, GameState ∗state)
- void destroy_game (GameState ∗state)
- void render_game_over (Camera ∗camera)
- void save_state (GameState ∗state)
- double landing_quality (Lander ∗lander)
- int calculate_score (Lander ∗state)

5.9.1 Typedef Documentation

5.9.1.1 GameState

```
typedef struct GameState GameState
```

5.9.2 Function Documentation

5.9.2.1 calculate_score()

```
int calculate_score (
             Lander * state )
```

5.9.2.2 destroy_game()

```
void destroy_game (
             GameState * state )
```

5.9.2.3 game_events()

```
Screen game_events (
             SDL_Event event,
             GameState * state )
```

5.9.2.4 init_game()

```
GameState init_game (
             SDL_Renderer * renderer,
             int * terrain_seed )
```

Sets up the default parameters for the lander, the camera and the world.

Parameters

renderer	Used for rendering the game

Returns

GameState containing the lander and camera structs

5.9.2.5 landing_quality()

```
double landing_quality (
            Lander * lander )
```

5.9.2.6 render_game_over()

```
void render_game_over (
            Camera * camera )
```

5.9.2.7 save_state()

```
void save_state (
            GameState * state )
```

5.9.2.8 update_game()

```
void update_game (
            GameState * state )
```

5.10 game.h

Go to the documentation of this file.

```
#ifndef GAME_H
#define GAME_H

#include <SDL.h>

#include "lander.h"
#include "camera.h"
#include "menu.h"

typedef struct GameState {
    Lander lander;
    Camera camera;
    double time_started;
    double delta_time;
    double game_over_dealy;
    bool game_over;
    bool successfull;
    bool destroyed;
    bool saved;
} GameState;

/// @brief Sets up the default parameters for the lander, the camera and the world.
/// @param renderer Used for rendering the game
/// @return GameState containing the lander and camera structs
GameState init_game(SDL_Renderer *renderer, int *terrain_seed);

void update_game(GameState *state);
Screen game_events(SDL_Event event, GameState *state);
void destroy_game(GameState *state);
void render_game_over(Camera *camera);
void save_state(GameState *state);
double landing_quality(Lander *lander);
int calculate_score(Lander *state);
#endif
```

5.11 include/lander.h File Reference

```
#include <SDL.h>
#include <stdbool.h>
#include "vector.h"
#include "camera.h"
#include "particle.h"
```

Data Structures

- struct Lander
- struct ImpactPoint

Typedefs

- typedef struct Lander Lander
- typedef struct ImpactPoint ImpactPoint

Enumerations

- enum Engine {
  MAIN_ENGINE , LEFT_ENGINE , RIGHT_ENGINE , ROTATE_CW ,
  ROTATE_CCW }

Functions

- Lander init_lander (SDL_Renderer ∗renderer)

  Initializes the lander struct's data.
- void destroy_lander (Lander ∗lander)

  Clean up memory after the game.
- void render_lander (Camera ∗camera, Lander ∗lander)

  Render the current frame of the lander.
- void display_dashboard (Camera ∗camera, Lander ∗lander)
- void update_lander (Lander ∗lander, double dt)

  Apply forces and update position and rotation of the lander.
- double get_lander_inertia (Lander ∗lander)

  Get the current inertia of the lander.
- double lander_total_mass (Lander ∗lander)

  Calculates the total mass based on remaining fuel.
- double get_torque (Vector2 point, Vector2 force)

  Calculates the torque from a force applied to a given point on the lander.
- Vector2 get_impact_force (Lander ∗lander, Vector2 point, double dt)

  Calculates the force applyed to the lander when colliding with the ground.
- void bulk_add_particles (Lander ∗lander, int count, double size, SDL_Rect area, double life, Vector2 velocity, double angle, SDL_Color start_color, SDL_Color end_color)
- Vector2 lander_to_world_coord (Lander ∗lander, Vector2 point)

5.11.1 Typedef Documentation

5.11.1.1 ImpactPoint

```
typedef struct ImpactPoint ImpactPoint
```

5.11.1.2 Lander

```
typedef struct Lander Lander
```

5.11.2 Enumeration Type Documentation

5.11.2.1 Engine

```
enum Engine
```

Enumerator

MAIN_ENGINE	
LEFT_ENGINE	
RIGHT_ENGINE	
ROTATE_CW	
ROTATE_CCW	

5.11.3 Function Documentation

5.11.3.1 bulk_add_particles()

```
void bulk_add_particles (
            Lander * lander,
            int count,
            double size,
            SDL_Rect area,
            double life,
            Vector2 velocity,
            double angle,
            SDL_Color start_color,
            SDL_Color end_color )
```

5.11.3.2 destroy_lander()

```
void destroy_lander (
            Lander * lander )
```

Clean up memory after the game.

Parameters

lander	The lander struct used in the game

5.11.3.3 display_dashboard()

```
void display_dashboard (
            Camera * camera,
            Lander * lander )
```

5.11.3.4 get_impact_force()

```
Vector2 get_impact_force (
            Lander * lander,
            Vector2 point,
            double dt )
```

Calculates the force applyed to the lander when colliding with the ground.

Parameters

lander	
point	Point of collision in pixels relative to the top left corner of the sprite
dt	Time since the last frame

Returns

Force applied to the lander by the ground at the given point, {0, 0} if that point is not colliding

5.11.3.5 get_lander_inertia()

```
double get_lander_inertia (
             Lander * lander )
```

Get the current inertia of the lander.

Parameters

lander	

Returns

Current inertia based on the lander's mass

5.11.3.6 get_torque()

```
double get_torque (
             Vector2 point,
             Vector2 force )
```

Calculates the torque from a force applied to a given point on the lander.

Parameters

point	The point to apply the force to, in pixels relative to the top left corner of the sprite
force	The force to apply

Returns

The resulting torque

5.11.3.7 init_lander()

```
Lander init_lander (
             SDL_Renderer * renderer )
```

Initializes the lander struct's data.

Parameters

renderer	Needed for loading sprites

Returns

Lander struct with default values set up

5.11.3.8 lander_to_world_coord()

```
Vector2 lander_to_world_coord (
             Lander * lander,
             Vector2 point )
```

5.11.3.9 lander_total_mass()

```
double lander_total_mass (
             Lander * lander )
```

Calculates the total mass based on remaining fuel.

Parameters

lander	

Returns

Total mass

5.11.3.10 render_lander()

```
void render_lander (
             Camera * camera,
             Lander * lander )
```

Render the current frame of the lander.

Parameters

camera	Camera to render with
lander	Lander to render

5.11.3.11 update_lander()

```
void update_lander (
             Lander * lander,
             double dt )
```

Apply forces and update position and rotation of the lander.

Parameters

lander	The lander to update
dt	Time since last frame

5.12 lander.h

Go to the documentation of this file.

```
#ifndef LANDER_H
#define LANDER_H

#include <SDL.h>
#include <stdbool.h>
#include "vector.h"
#include "camera.h"
#include "particle.h"

typedef enum {
    MAIN_ENGINE,
    LEFT_ENGINE,
    RIGHT_ENGINE,
    ROTATE_CW,
    ROTATE_CCW,
} Engine;

typedef struct Lander {
    Vector2 position;
    Vector2 velocity;
    double rotation;
    double angular_velocity;
    double dry_mass;
    double propellant;
    int impact_count;
    bool engines[5];
    List particle_system;
} Lander;

typedef struct ImpactPoint {
    Vector2 point;
    bool can_collide;
} ImpactPoint;

/// @brief Initializes the lander struct's data.
/// @param renderer Needed for loading sprites
/// @return Lander struct with default values set up
Lander init_lander(SDL_Renderer *renderer);

/// @brief Clean up memory after the game.
/// @param lander The lander struct used in the game
void destroy_lander(Lander *lander);

/// @brief Render the current frame of the lander.
/// @param camera Camera to render with
/// @param lander Lander to render
void render_lander(Camera *camera, Lander *lander);

void display_dashboard(Camera *camera, Lander *lander);

/// @brief Apply forces and update position and rotation of the lander.
/// @param lander The lander to update
/// @param dt Time since last frame
void update_lander(Lander *lander, double dt);

/// @brief Get the current inertia of the lander.
/// @param lander
/// @return Current inertia based on the lander's mass
double get_lander_inertia(Lander *lander);

/// @brief Calculates the total mass based on remaining fuel
/// @param lander
/// @return Total mass
double lander_total_mass(Lander *lander);

/// @brief Calculates the torque from a force applied to a given point on the lander.
/// @param point The point to apply the force to, in pixels relative to the top left corner of the sprite
/// @param force The force to apply
/// @return The resulting torque
double get_torque(Vector2 point, Vector2 force);

/// @brief Calculates the force applyed to the lander when colliding with the ground.
/// @param lander
/// @param point Point of collision in pixels relative to the top left corner of the sprite
/// @param dt Time since the last frame
/// @return Force applied to the lander by the ground at the given point, {0, 0} if that point is not
    colliding
Vector2 get_impact_force(Lander *lander, Vector2 point, double dt);
void bulk_add_particles(Lander *lander, int count, double size, SDL_Rect area, double life, Vector2
    velocity, double angle, SDL_Color start_color, SDL_Color end_color);
Vector2 lander_to_world_coord(Lander *lander, Vector2 point);
#endif
```

5.13 include/leaderboard.h File Reference

`#include <SDL.h>`

Functions

- void init_leaderboard ()
- void render_leaderboard (SDL_Renderer *renderer)
- void destroy_leaderboard ()

5.13.1 Function Documentation

5.13.1.1 destroy_leaderboard()

`void destroy_leaderboard ( )`

5.13.1.2 init_leaderboard()

`void init_leaderboard ( )`

5.13.1.3 render_leaderboard()

```
void render_leaderboard (
              SDL_Renderer * renderer )
```

5.14 leaderboard.h

Go to the documentation of this file.

```
#ifndef LEADERBOARD_H
#define LEADERBOARD_H

#include <SDL.h>

void init_leaderboard();
void render_leaderboard(SDL_Renderer *renderer);
void destroy_leaderboard();

#endif
```

5.15 include/main.h File Reference

Typedefs

- typedef enum Screen Screen

  The available screens to choose from in the menu.

Enumerations

- enum Screen {
  MENU , GAME , LEADERBOARD , MENU ,
  GAME , LEADERBOARD }

  The available screens to choose from in the menu.

5.15.1 Typedef Documentation

5.15.1.1 Screen

```
typedef enum Screen Screen
```

The available screens to choose from in the menu.

5.15.2 Enumeration Type Documentation

5.15.2.1 Screen

```
enum Screen
```

The available screens to choose from in the menu.

Enumerator

MENU	
GAME	
LEADERBOARD	
MENU	
GAME	
LEADERBOARD	

5.16 main.h

Go to the documentation of this file.

```
#ifndef MAIN_H
#define MAIN_H

/// @brief The available screens to choose from in the menu
typedef enum Screen {
    MENU,
    GAME,
    LEADERBOARD
} Screen;

#endif
```

5.17 include/menu.h File Reference

```
#include <SDL.h>
#include <SDL_ttf.h>
```

Typedefs

- typedef enum Screen Screen

Enumerations

- enum Screen {
  MENU , GAME , LEADERBOARD , MENU ,
  GAME , LEADERBOARD }

Functions

- void init_menu ()
- void destroy_menu ()
- void render_menu (SDL_Renderer *renderer)
- Screen menu_events (SDL_Event event)

5.17.1 Typedef Documentation

5.17.1.1 Screen

```
typedef enum Screen Screen
```

5.17.2 Enumeration Type Documentation

5.17.2.1 Screen

```
enum Screen
```

Enumerator

MENU	
GAME	
LEADERBOARD	
MENU	
GAME	
LEADERBOARD	

5.17.3 Function Documentation

5.17.3.1 destroy_menu()

```
void destroy_menu ( )
```

5.17.3.2 init_menu()

```
void init_menu ( )
```

5.17.3.3 menu_events()

```
Screen menu_events (
            SDL_Event event )
```

5.17.3.4 render_menu()

```
void render_menu (
             SDL_Renderer * renderer )
```

5.18 menu.h

Go to the documentation of this file.

```
#ifndef MENU_H
#define MENU_H

#include <SDL.h>
#include <SDL_ttf.h>

typedef enum Screen {
    MENU,
    GAME,
    LEADERBOARD
} Screen;

void init_menu();
void destroy_menu();
void render_menu(SDL_Renderer *renderer);
Screen menu_events(SDL_Event event);
#endif
```

5.19 include/particle.h File Reference

```
#include "vector.h"
#include "camera.h"
```

Data Structures

- struct Particle
- struct ListElement
- struct List

Typedefs

- typedef struct Particle Particle
- typedef struct ListElement ListElement
- typedef struct List List

Functions

- void append_particle (List ∗list, Particle p)
- void delete_particle (List ∗list, ListElement ∗particle)
- void update_particles (List ∗list, double dt)
- void render_particles (Camera ∗camera, List ∗list)
- void destroy_particles (List ∗list)

5.19.1 Typedef Documentation

5.19.1.1 List

```
typedef struct List List
```

5.19.1.2 ListElement

```
typedef struct ListElement ListElement
```

5.19.1.3 Particle

```
typedef struct Particle Particle
```

5.19.2 Function Documentation

5.19.2.1 append_particle()

```
void append_particle (
            List * list,
            Particle p )
```

5.19.2.2 delete_particle()

```
void delete_particle (
            List * list,
            ListElement * particle )
```

5.19.2.3 destroy_particles()

```
void destroy_particles (
            List * list )
```

5.19.2.4 render_particles()

```
void render_particles (
             Camera * camera,
             List * list )
```

5.19.2.5 update_particles()

```
void update_particles (
             List * list,
             double dt )
```

5.20 particle.h

Go to the documentation of this file.

```
#ifndef PARTICLE_H
#define PARTICLE_H

#include "vector.h"
#include "camera.h"

typedef struct Particle {
    SDL_Color start_color;
    SDL_Color end_color;
    Vector2 velocity;
    Vector2 position;
    double life_time;
    double lived;
    double size;
} Particle;

typedef struct ListElement ListElement;

struct ListElement {
    Particle particle;
    ListElement *next;
    ListElement *prev;
};

typedef struct List {
    ListElement *first;
    ListElement *last;
} List;


void append_particle(List *list, Particle p);
void delete_particle(List *list, ListElement *particle);
void update_particles(List *list, double dt);
void render_particles(Camera *camera, List *list);
void destroy_particles(List *list);

#endif
```

5.21 include/terrain.h File Reference

```
#include "camera.h"
```

Functions

- double get_terrain_height (double x)

    Calculates the height of the terrain at a given position.
- void render_terrain (Camera ∗camera)

    Renders the currently visible part of the terrain.
- void init_terrain (int ∗set_seed)

Variables

- int TERRAIN_SEED

5.21.1 Function Documentation

5.21.1.1 get_terrain_height()

```
double get_terrain_height (
             double x )
```

Calculates the height of the terrain at a given position.

Parameters

x	The x coordinate we want the height at in world space

Returns

Height of the terrain in meters at x

5.21.1.2 init_terrain()

```
void init_terrain (
             int ∗ set_seed )
```

5.21.1.3 render_terrain()

```
void render_terrain (
             Camera ∗ camera )
```

Renders the currently visible part of the terrain.

Parameters

camera	The camera to render with

5.21.2 Variable Documentation

5.21.2.1 TERRAIN_SEED

```
int TERRAIN_SEED [extern]
```

5.22 terrain.h

Go to the documentation of this file.

```
#ifndef TERRAIN_H
#define TERRAIN_H

#include "camera.h"

extern int TERRAIN_SEED;

/// @brief Calculates the height of the terrain at a given position.
/// @param x The x coordinate we want the height at in world space
/// @return Height of the terrain in meters at x
double get_terrain_height(double x);

/// @brief Renders the currently visible part of the terrain.
/// @param camera The camera to render with
void render_terrain(Camera *camera);

void init_terrain(int *set_seed);
#endif
```

5.23 include/text_io.h File Reference

```
#include <stdbool.h>
#include <SDL.h>
#include <SDL_ttf.h>
```

Functions

- bool input_text (char ∗dest, size_t hossz, SDL_Rect teglalap, SDL_Color hatter, SDL_Color szoveg, TTF_↩ Font ∗font, SDL_Renderer ∗renderer)

    From INFOC.
- SDL_Rect render_text_centered (SDL_Renderer ∗renderer, SDL_Rect ∗container, char ∗text, TTF_Font ∗font, SDL_Color text_color, double y_offset)

5.23.1 Function Documentation

5.23.1.1 input_text()

```
bool input_text (
            char * dest,
            size_t hossz,
            SDL_Rect teglalap,
            SDL_Color hatter,
            SDL_Color szoveg,
            TTF_Font * font,
            SDL_Renderer * renderer )
```

From INFOC.

5.23.1.2 render_text_centered()

```
SDL_Rect render_text_centered (
            SDL_Renderer * renderer,
            SDL_Rect * container,
            char * text,
            TTF_Font * font,
            SDL_Color text_color,
            double y_offset )
```

5.24 text_io.h

Go to the documentation of this file.

```
#ifndef TEXTIO_H
#define TEXTIO_H

#include <stdbool.h>
#include <SDL.h>
#include <SDL_ttf.h>

bool input_text(char *dest, size_t hossz, SDL_Rect teglalap, SDL_Color hatter, SDL_Color szoveg, TTF_Font
    *font, SDL_Renderer *renderer);
SDL_Rect render_text_centered(SDL_Renderer *renderer, SDL_Rect *container, char *text, TTF_Font *font,
    SDL_Color text_color, double y_offset);
#endif
```

5.25 include/vector.h File Reference

```
#include <SDL.h>
```

Data Structures

- struct Vector2

Typedefs

- typedef struct Vector2 Vector2

Functions

- Vector2 V_add (Vector2 v1, Vector2 v2)

  Adds two vectors together.
- Vector2 V_subtract (Vector2 v1, Vector2 v2)

  Suptracts two vectors from each other.
- Vector2 V_multiply_const (Vector2 v, double c)

  Multiplies a vector by a constant value.
- Vector2 V_multiply (Vector2 v1, Vector2 v2)

  Multiplies two vectors together.
- Vector2 V_divide_const (Vector2 v, double c)

  Divides a vector by a constant value.
- Vector2 V_normalize (Vector2 v)

  Calculates the vector with the same direction as the original, but with a length of 1.
- Vector2 V_rotate (Vector2 v, double deg)

  Rotates the vector.
- double V_len (Vector2 v)

  Calculates the length of a vector.
- double V_cross_product (Vector2 v1, Vector2 v2)

  Calculates the cross product of two vectors.
- SDL_Point V_to_point (Vector2 v)

  Converts a Vector2 to an SDL_Point.

5.25.1 Typedef Documentation

5.25.1.1 Vector2

```
typedef struct Vector2 Vector2
```

5.25.2 Function Documentation

5.25.2.1 V_add()

```
Vector2 V_add (
                Vector2 v1,
                Vector2 v2 )
```

Adds two vectors together.

5.25.2.2 V_cross_product()

```
double V_cross_product (
            Vector2 v1,
            Vector2 v2 )
```

Calculates the cross product of two vectors.

5.25.2.3 V_divide_const()

```
Vector2 V_divide_const (
            Vector2 v,
            double c )
```

Divides a vector by a constant value.

5.25.2.4 V_len()

```
double V_len (
            Vector2 v )
```

Calculates the length of a vector.

5.25.2.5 V_multiply()

```
Vector2 V_multiply (
            Vector2 v1,
            Vector2 v2 )
```

Multiplies two vectors together.

5.25.2.6 V_multiply_const()

```
Vector2 V_multiply_const (
            Vector2 v,
            double c )
```

Multiplies a vector by a constant value.

5.25.2.7 V_normalize()

```
Vector2 V_normalize (
            Vector2 v )
```

Calculates the vector with the same direction as the original, but with a length of 1.

5.25.2.8 V_rotate()

```
Vector2 V_rotate (
            Vector2 v,
            double deg )
```

Rotates the vector.

Parameters

v	
deg	Amount to rotate by in degrees

5.25.2.9 V_subtract()

```
Vector2 V_subtract (
            Vector2 v1,
            Vector2 v2 )
```

Suptracts two vectors from each other.

5.25.2.10 V_to_point()

```
SDL_Point V_to_point (
            Vector2 v )
```

Converts a Vector2 to an SDL_Point.

5.26 vector.h

Go to the documentation of this file.

```
#ifndef VECTOR_H
#define VECTOR_H

#include <SDL.h>

typedef struct Vector2 {
    double x;
    double y;
} Vector2;

/// @brief Adds two vectors together.
Vector2 V_add(Vector2 v1, Vector2 v2);

/// @brief Suptracts two vectors from each other.
Vector2 V_subtract(Vector2 v1, Vector2 v2);

/// @brief Multiplies a vector by a constant value.
Vector2 V_multiply_const(Vector2 v, double c);

/// @brief Multiplies two vectors together.
Vector2 V_multiply(Vector2 v1, Vector2 v2);

/// @brief Divides a vector by a constant value.
Vector2 V_divide_const(Vector2 v, double c);

/// @brief Calculates the vector with the same direction as the original, but with a length of 1.
Vector2 V_normalize(Vector2 v);

/// @brief Rotates the vector.
/// @param v
/// @param deg Amount to rotate by in degrees
Vector2 V_rotate(Vector2 v, double deg);

/// @brief Calculates the length of a vector.
double V_len(Vector2 v);

/// @brief Calculates the cross product of two vectors
double V_cross_product(Vector2 v1, Vector2 v2);

/// @brief Converts a Vector2 to an SDL_Point
SDL_Point V_to_point(Vector2 v);

#endif
```

5.27 README.md File Reference

5.28 src/button.c File Reference

```
#include <SDL.h>
#include <SDL2_gfxPrimitives.h>
#include <SDL_ttf.h>
#include "button.h"
```

Functions

- void render_button (SDL_Renderer ∗renderer, TTF_Font ∗font, Button ∗button)

Variables

- const SDL_Color bg_color = {129, 151, 150, 255}
- const SDL_Color hover_color = {168, 202, 88, 255}
- const SDL_Color fg_color = {255, 255, 255, 255}

5.28.1 Function Documentation

5.28.1.1 render_button()

```
void render_button (
            SDL_Renderer * renderer,
            TTF_Font * font,
            Button * button )
```

5.28.2 Variable Documentation

5.28.2.1 bg_color

```
const SDL_Color bg_color = {129, 151, 150, 255}
```

5.28.2.2 fg_color

```
const SDL_Color fg_color = {255, 255, 255, 255}
```

5.28.2.3 hover_color

```
const SDL_Color hover_color = {168, 202, 88, 255}
```

5.29 src/camera.c File Reference

```
#include "game.h"
#include "vector.h"
```

Functions

- double lerp (double a, double b, double t)

  Function to interpolate between two points.
- Vector2 lerp2 (Vector2 a, Vector2 b, double t)
- void update_camera (Camera ∗camera, Vector2 lander_pos, double dt)

  Updates the position and zoom level of the camera.
- Vector2 get_world_coordinates (Camera ∗camera, Vector2 screen_coordinates)

  Convert screen coordinates to world coordinates.
- Vector2 get_screen_coordinates (Camera ∗camera, Vector2 world_coordinates)

  Convert world coordinates to screen coordinates.

Variables

- const double PIXELS_PER_METER = 7

    Used for converting between screen space and world space.
- const int camera_speed = 5

5.29.1 Function Documentation

5.29.1.1 get_screen_coordinates()

```
Vector2 get_screen_coordinates (
            Camera * camera,
            Vector2 world_coordinates )
```

Convert world coordinates to screen coordinates.

Parameters

camera	The camera struct used for calculations
world_coordinates	The point to be converted

Returns

The coordinates of the point in screen space

5.29.1.2 get_world_coordinates()

```
Vector2 get_world_coordinates (
            Camera * camera,
            Vector2 screen_coordinates )
```

Convert screen coordinates to world coordinates.

Parameters

camera	The camera struct used for calculations
screen_coordinates	the point to bo converted

Returns

The coordinates of the point in world space

5.29.1.3 lerp()

```
double lerp (
             double a,
             double b,
             double t )
```

Function to interpolate between two points.

Parameters

a	point a
b	point b
t	time (0-1)

Returns

a if t = 0 b if t = 1

5.29.1.4 lerp2()

```
Vector2 lerp2 (
             Vector2 a,
             Vector2 b,
             double t )
```

5.29.1.5 update_camera()

```
void update_camera (
             Camera * camera,
             Vector2 lander_pos,
             double dt )
```

Updates the position and zoom level of the camera.

Parameters

camera	The camera struct to update
lander_pos	Positoin of the lander for tracking
dt	Time in seconds since the last frame

5.29.2 Variable Documentation

5.29.2.1 camera_speed

```
const int camera_speed = 5
```

5.29.2.2 PIXELS_PER_METER

```
const double PIXELS_PER_METER = 7
```

Used for converting between screen space and world space.

5.30 src/file_handler.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "file_handler.h"
#include "debugmalloc.h"
```

Functions

- void append_score (Score *score)
- int read_scores (Score **scores)

5.30.1 Function Documentation

5.30.1.1 append_score()

```
void append_score (
            Score * score )
```

5.30.1.2 read_scores()

```
int read_scores (
            Score ** scores )
```

5.31 src/game.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <SDL.h>
#include <SDL_image.h>
#include <SDL2_gfxPrimitives.h>
#include "game.h"
#include "terrain.h"
#include "vector.h"
#include "lander.h"
#include "camera.h"
#include "events.h"
#include "menu.h"
#include "button.h"
#include "particle.h"
#include "text_io.h"
#include "file_handler.h"
```

Functions

- GameState init_game (SDL_Renderer ∗renderer, int ∗terrain_seed)

  Sets up the default parameters for the lander, the camera and the world.
- void update_game (GameState ∗state)
- void save_state (GameState ∗state)
- void render_game_over (Camera ∗camera)
- Screen game_events (SDL_Event event, GameState ∗state)
- int calculate_score (Lander ∗lander)
- double landing_quality (Lander ∗lander)
- void destroy_game (GameState ∗state)

Variables

- TTF_Font ∗ font_large
- TTF_Font ∗ font_small

5.31.1 Function Documentation

5.31.1.1 calculate_score()

```
int calculate_score (
            Lander ∗ lander )
```

5.31.1.2 destroy_game()

```
void destroy_game (
            GameState * state )
```

5.31.1.3 game_events()

```
Screen game_events (
            SDL_Event event,
            GameState * state )
```

5.31.1.4 init_game()

```
GameState init_game (
            SDL_Renderer * renderer,
            int * terrain_seed )
```

Sets up the default parameters for the lander, the camera and the world.

Parameters

renderer	Used for rendering the game

Returns

GameState containing the lander and camera structs

5.31.1.5 landing_quality()

```
double landing_quality (
            Lander * lander )
```

5.31.1.6 render_game_over()

```
void render_game_over (
            Camera * camera )
```

5.31.1.7 save_state()

```
void save_state (
            GameState * state )
```

5.31.1.8 update_game()

```
void update_game (
            GameState * state )
```

5.31.2 Variable Documentation

5.31.2.1 font_large

```
TTF_Font* font_large
```

5.31.2.2 font_small

```
TTF_Font* font_small
```

5.32 src/lander.c File Reference

```
#include <SDL.h>
#include <SDL_image.h>
#include <SDL2_gfxPrimitives.h>
#include <stdbool.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "lander.h"
#include "vector.h"
#include "camera.h"
#include "terrain.h"
#include "events.h"
#include "particle.h"
```

Functions

- Lander init_lander (SDL_Renderer ∗renderer)

    Initializes the lander struct's data.
- void destroy_lander (Lander ∗lander)

    Clean up memory after the game.
- void render_lander (Camera ∗camera, Lander ∗lander)

    Render the current frame of the lander.
- void display_dashboard (Camera ∗camera, Lander ∗lander)
- Vector2 lander_to_world_coord (Lander ∗lander, Vector2 point)
- void bulk_add_particles (Lander ∗lander, int count, double size, SDL_Rect area, double life, Vector2 velocity, double angle, SDL_Color start_color, SDL_Color end_color)
- void update_lander (Lander ∗lander, double dt)

    Apply forces and update position and rotation of the lander.
- double lander_total_mass (Lander ∗lander)

    Calculates the total mass based on remaining fuel.
- double get_lander_inertia (Lander ∗lander)

    Get the current inertia of the lander.
- Vector2 to_metric (Vector2 point)
- double get_torque (Vector2 point, Vector2 force)

    Calculates the torque from a force applied to a given point on the lander.
- Vector2 get_impact_force (Lander ∗lander, Vector2 point, double dt)

    Calculates the force applyed to the lander when colliding with the ground.

Variables

- const int dry_mass = 7000
- const int propellant_mass = 8200
- const int inertia_min = 800000
- const int inertia_max = 1000000
- const int size_px = 64
- const int main_engine_thrust = 45040
- const int rcs_thrust = 20000
- const int main_engine_fuel_rate = 136
- const int rcs_fuel_rate = 13
- const double g = 1.62
- const double friction_coefficient = 0.2
- const Vector2 center_of_mass = {32.5, 20}
- SDL_Texture ∗ lander_texture
- SDL_Texture ∗ dashboard_texture

5.32.1 Function Documentation

5.32.1.1 bulk_add_particles()

```
void bulk_add_particles (
             Lander * lander,
             int count,
             double size,
             SDL_Rect area,
             double life,
             Vector2 velocity,
             double angle,
             SDL_Color start_color,
             SDL_Color end_color )
```

5.32.1.2 destroy_lander()

```
void destroy_lander (
             Lander * lander )
```

Clean up memory after the game.

Parameters

lander	The lander struct used in the game

5.32.1.3 display_dashboard()

```
void display_dashboard (
             Camera * camera,
             Lander * lander )
```

5.32.1.4 get_impact_force()

```
Vector2 get_impact_force (
             Lander * lander,
             Vector2 point,
             double dt )
```

Calculates the force applyed to the lander when colliding with the ground.

Parameters

lander	
point	Point of collision in pixels relative to the top left corner of the sprite
dt	Time since the last frame

Returns

Force applied to the lander by the ground at the given point, {0, 0} if that point is not colliding

5.32.1.5 get_lander_inertia()

```
double get_lander_inertia (
            Lander * lander )
```

Get the current inertia of the lander.

Parameters

lander	

Returns

Current inertia based on the lander's mass

5.32.1.6 get_torque()

```
double get_torque (
            Vector2 point,
            Vector2 force )
```

Calculates the torque from a force applied to a given point on the lander.

Parameters

point	The point to apply the force to, in pixels relative to the top left corner of the sprite
force	The force to apply

Returns

The resulting torque

5.32.1.7 init_lander()

```
Lander init_lander (
            SDL_Renderer * renderer )
```

Initializes the lander struct's data.

Parameters

renderer	Needed for loading sprites

Returns

Lander struct with default values set up

5.32.1.8 lander_to_world_coord()

```
Vector2 lander_to_world_coord (
            Lander * lander,
            Vector2 point )
```

5.32.1.9 lander_total_mass()

```
double lander_total_mass (
            Lander * lander )
```

Calculates the total mass based on remaining fuel.

Parameters

lander	

Returns

Total mass

5.32.1.10 render_lander()

```
void render_lander (
            Camera * camera,
            Lander * lander )
```

Render the current frame of the lander.

Parameters

camera	Camera to render with
lander	Lander to render

5.32.1.11 to_metric()

```
Vector2 to_metric (
             Vector2 point )
```

5.32.1.12 update_lander()

```
void update_lander (
             Lander * lander,
             double dt )
```

Apply forces and update position and rotation of the lander.

Parameters

lander	The lander to update
dt	Time since last frame

5.32.2 Variable Documentation

5.32.2.1 center_of_mass

```
const Vector2 center_of_mass = {32.5, 20}
```

5.32.2.2 dashboard_texture

```
SDL_Texture* dashboard_texture
```

5.32.2.3 dry_mass

```
const int dry_mass = 7000
```

5.32.2.4 friction_coefficient

```
const double friction_coefficient = 0.2
```

5.32.2.5 g

```
const double g = 1.62
```

5.32.2.6 inertia_max

```
const int inertia_max = 1000000
```

5.32.2.7 inertia_min

```
const int inertia_min = 800000
```

5.32.2.8 lander_texture

```
SDL_Texture* lander_texture
```

5.32.2.9 main_engine_fuel_rate

```
const int main_engine_fuel_rate = 136
```

5.32.2.10 main_engine_thrust

```
const int main_engine_thrust = 45040
```

5.32.2.11 propellant_mass

```
const int propellant_mass = 8200
```

5.32.2.12 rcs_fuel_rate

```
const int rcs_fuel_rate = 13
```

5.32.2.13 rcs_thrust

```
const int rcs_thrust = 20000
```

5.32.2.14 size_px

```
const int size_px = 64
```

5.33 src/leaderboard.c File Reference

```
#include <SDL.h>
#include <SDL_ttf.h>
#include "leaderboard.h"
#include "file_handler.h"
#include "text_io.h"
#include "debugmalloc.h"
```

Functions

- void init_leaderboard ()
- void render_leaderboard (SDL_Renderer ∗renderer)
- void destroy_leaderboard ()

Variables

- Score ∗ score = NULL
- int count = 0
- TTF_Font ∗ font

5.33.1 Function Documentation

5.33.1.1 destroy_leaderboard()

```
void destroy_leaderboard ( )
```

5.33.1.2 init_leaderboard()

```
void init_leaderboard ( )
```

5.33.1.3 render_leaderboard()

```
void render_leaderboard (
             SDL_Renderer * renderer )
```

5.33.2 Variable Documentation

5.33.2.1 count

```
int count = 0
```

5.33.2.2 font

```
TTF_Font* font
```

5.33.2.3 score

```
Score* score = NULL
```

5.34 src/main.c File Reference

```
#include <stdlib.h>
#include <stdbool.h>
#include <SDL.h>
#include <SDL_ttf.h>
#include "game.h"
#include "menu.h"
#include "leaderboard.h"
#include "debugmalloc.h"
```

Functions

- int main (int argc, char *argv[ ])

5.34.1 Function Documentation

5.34.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

5.35 src/menu.c File Reference

```
#include <SDL.h>
#include <SDL_ttf.h>
#include "menu.h"
#include "button.h"
```

Functions

- void init_menu ()
- void destroy_menu ()
- void render_menu (SDL_Renderer *renderer)
- Screen menu_events (SDL_Event event)

Variables

- Button buttons [ ]
- const int button_count = 2
- const int margin = 20
- TTF_Font * font

5.35.1 Function Documentation

5.35.1.1 destroy_menu()

```
void destroy_menu ( )
```

5.35.1.2 init_menu()

```
void init_menu ( )
```

5.35.1.3 menu_events()

```
Screen menu_events (
             SDL_Event event )
```

5.35.1.4 render_menu()

```
void render_menu (
             SDL_Renderer * renderer )
```

5.35.2 Variable Documentation

5.35.2.1 button_count

```
const int button_count = 2
```

5.35.2.2 buttons

```
Button buttons[ ]
```

Initial value:

```
= {
    {
        .text = "New Game",
    },
    {
        .text = "Leaderboard"
    }
}
```

5.35.2.3 font

```
TTF_Font* font
```

5.35.2.4 margin

```
const int margin = 20
```

5.36 src/particle.c File Reference

```
#include <SDL.h>
#include <SDL2_gfxPrimitives.h>
#include <stdlib.h>
#include <math.h>
#include "particle.h"
#include "vector.h"
#include "camera.h"
#include "terrain.h"
#include "debugmalloc.h"
```

Functions

- void append_particle (List ∗list, Particle p)
- void update_particles (List ∗list, double dt)
- void render_particles (Camera ∗camera, List ∗list)
- void delete_particle (List ∗list, ListElement ∗particle)
- void destroy_particles (List ∗list)

5.36.1 Function Documentation

5.36.1.1 append_particle()

```
void append_particle (
            List * list,
            Particle p )
```

5.36.1.2 delete_particle()

```
void delete_particle (
            List * list,
            ListElement * particle )
```

5.36.1.3 destroy_particles()

```
void destroy_particles (
            List * list )
```

5.36.1.4 render_particles()

```
void render_particles (
            Camera * camera,
            List * list )
```

5.36.1.5 update_particles()

```
void update_particles (
            List * list,
            double dt )
```

5.37 src/terrain.c File Reference

```
#include <SDL.h>
#include <SDL2_gfxPrimitives.h>
#include <math.h>
#include <stdlib.h>
#include "camera.h"
#include "vector.h"
```

Functions

- void init_terrain (int ∗set_seed)
- double pseudo_random (int x)
- double noise (double x, int scale)
- double get_terrain_height (double x)

    Calculates the height of the terrain at a given position.
- void render_terrain (Camera ∗camera)

    Renders the currently visible part of the terrain.

Variables

- const int terrain_max_height = 50
- int TERRAIN_SEED

5.37.1 Function Documentation

5.37.1.1 get_terrain_height()

```
double get_terrain_height (
            double x )
```

Calculates the height of the terrain at a given position.

Parameters

x	The x coordinate we want the height at in world space

Returns

Height of the terrain in meters at x

5.37.1.2 init_terrain()

```
void init_terrain (
             int * set_seed )
```

5.37.1.3 noise()

```
double noise (
             double x,
             int scale )
```

5.37.1.4 pseudo_random()

```
double pseudo_random (
             int x )
```

5.37.1.5 render_terrain()

```
void render_terrain (
             Camera * camera )
```

Renders the currently visible part of the terrain.

Parameters

camera	The camera to render with

5.37.2 Variable Documentation

5.37.2.1 terrain_max_height

```
const int terrain_max_height = 50
```

5.37.2.2 TERRAIN_SEED

```
int TERRAIN_SEED
```

5.38 src/text_io.c File Reference

```
#include <SDL.h>
#include <SDL2_gfxPrimitives.h>
#include <SDL_ttf.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include "text_io.h"
```

Functions

- bool valid (char ∗text)
- bool input_text (char ∗dest, size_t hossz, SDL_Rect teglalap, SDL_Color hatter, SDL_Color szoveg, TTF_↩ Font ∗font, SDL_Renderer ∗renderer)

    From INFOC.
- SDL_Rect render_text_centered (SDL_Renderer ∗renderer, SDL_Rect ∗container, char ∗text, TTF_Font ∗font, SDL_Color text_color, double y_offset)

5.38.1 Function Documentation

5.38.1.1 input_text()

```
bool input_text (
            char * dest,
            size_t hossz,
            SDL_Rect teglalap,
            SDL_Color hatter,
            SDL_Color szoveg,
            TTF_Font * font,
            SDL_Renderer * renderer )
```

From INFOC.

5.38.1.2 render_text_centered()

```
SDL_Rect render_text_centered (
             SDL_Renderer * renderer,
             SDL_Rect * container,
             char * text,
             TTF_Font * font,
             SDL_Color text_color,
             double y_offset )
```

5.38.1.3 valid()

```
bool valid (
             char * text )
```

5.39 src/vector.c File Reference

```
#include <math.h>
#include <SDL.h>
#include "vector.h"
```

Functions

- Vector2 V_add (Vector2 v1, Vector2 v2)

  Adds two vectors together.
- Vector2 V_subtract (Vector2 v1, Vector2 v2)

  Suptracts two vectors from each other.
- Vector2 V_multiply_const (Vector2 v, double c)

  Multiplies a vector by a constant value.
- Vector2 V_multiply (Vector2 v1, Vector2 v2)

  Multiplies two vectors together.
- Vector2 V_divide_const (Vector2 v, double c)

  Divides a vector by a constant value.
- double V_len (Vector2 v)

  Calculates the length of a vector.
- Vector2 V_normalize (Vector2 v)

  Calculates the vector with the same direction as the original, but with a length of 1.
- double V_cross_product (Vector2 v1, Vector2 v2)

  Calculates the cross product of two vectors.
- SDL_Point V_to_point (Vector2 v)

  Converts a Vector2 to an SDL_Point.
- Vector2 V_rotate (Vector2 v, double deg)

  Rotates the vector.

5.39.1 Function Documentation

5.39.1.1 V_add()

```
Vector2 V_add (
            Vector2 v1,
            Vector2 v2 )
```

Adds two vectors together.

5.39.1.2 V_cross_product()

```
double V_cross_product (
            Vector2 v1,
            Vector2 v2 )
```

Calculates the cross product of two vectors.

5.39.1.3 V_divide_const()

```
Vector2 V_divide_const (
            Vector2 v,
            double c )
```

Divides a vector by a constant value.

5.39.1.4 V_len()

```
double V_len (
            Vector2 v )
```

Calculates the length of a vector.

5.39.1.5 V_multiply()

```
Vector2 V_multiply (
            Vector2 v1,
            Vector2 v2 )
```

Multiplies two vectors together.

5.39.1.6 V_multiply_const()

```
Vector2 V_multiply_const (
            Vector2 v,
            double c )
```

Multiplies a vector by a constant value.

5.39.1.7 V_normalize()

```
Vector2 V_normalize (
            Vector2 v )
```

Calculates the vector with the same direction as the original, but with a length of 1.

5.39.1.8 V_rotate()

```
Vector2 V_rotate (
            Vector2 v,
            double deg )
```

Rotates the vector.

Parameters

v	
deg	Amount to rotate by in degrees

5.39.1.9 V_subtract()

```
Vector2 V_subtract (
            Vector2 v1,
            Vector2 v2 )
```

Suptracts two vectors from each other.

5.39.1.10 V_to_point()

```
SDL_Point V_to_point (
            Vector2 v )
```

Converts a Vector2 to an SDL_Point.

Index

angular_velocity
    Lander, 12
append_particle
    particle.c, 66
    particle.h, 40
append_score
    file_handler.c, 52
    file_handler.h, 25

bg_color
    button.c, 49
bulk_add_particles
    lander.c, 56
    lander.h, 30
Button, 7
    button.h, 19
    hover, 7
    rect, 7
    text, 7
button.c
    bg_color, 49
    fg_color, 49
    hover_color, 49
    render_button, 49
button.h
    Button, 19
    render_button, 20
button_count
    menu.c, 65
buttons
    menu.c, 65

calculate_score
    game.c, 53
    game.h, 27
Camera, 8
    camera.h, 21
    height, 8
    position, 8
    renderer, 8
    width, 9
    zoom, 9
camera
    GameState, 9
camera.c
    camera_speed, 51
    get_screen_coordinates, 50
    get_world_coordinates, 50
    lerp, 50
    lerp2, 51
    PIXELS_PER_METER, 52
    update_camera, 51
camera.h
    Camera, 21
    get_screen_coordinates, 21
    get_world_coordinates, 22
    lerp, 22
    PIXELS_PER_METER, 23
    update_camera, 22
camera_speed
    camera.c, 51
can_collide
    ImpactPoint, 11
center_of_mass
    lander.c, 60
count
    leaderboard.c, 63

dashboard_texture
    lander.c, 60
DEATH_EVENT_CODE
    events.h, 24
delete_particle
    particle.c, 66
    particle.h, 40
delta_time
    GameState, 9
destroy_game
    game.c, 53
    game.h, 27
destroy_lander
    lander.c, 57
    lander.h, 31
destroy_leaderboard
    leaderboard.c, 62
    leaderboard.h, 35
destroy_menu
    menu.c, 64
    menu.h, 38
destroy_particles
    particle.c, 66
    particle.h, 40
destroyed
    GameState, 10
display_dashboard
    lander.c, 57
    lander.h, 31
dry_mass
    Lander, 12
    lander.c, 60

end_color
  Particle, 15
Engine
  lander.h, 30
engines
  Lander, 12
EventCode
  events.h, 24
events.h
  DEATH_EVENT_CODE, 24
  EventCode, 24
  SUCCESS_EVENT_CODE, 24

fg_color
  button.c, 49
file_handler.c
  append_score, 52
  read_scores, 52
file_handler.h
  append_score, 25
  read_scores, 25
  Score, 25
first
  List, 13
font
  leaderboard.c, 63
  menu.c, 65
font_large
  game.c, 55
font_small
  game.c, 55
friction_coefficient
  lander.c, 60
fuel
  Score, 16

g
  lander.c, 61
GAME
  main.h, 37
  menu.h, 38
game.c
  calculate_score, 53
  destroy_game, 53
  font_large, 55
  font_small, 55
  game_events, 54
  init_game, 54
  landing_quality, 54
  render_game_over, 54
  save_state, 54
  update_game, 55
game.h
  calculate_score, 27
  destroy_game, 27
  game_events, 27
  GameState, 27
  init_game, 27
  landing_quality, 28
  render_game_over, 28
  save_state, 28
  update_game, 28
game_events
  game.c, 54
  game.h, 27
game_over
  GameState, 10
game_over_dealy
  GameState, 10
GameState, 9
  camera, 9
  delta_time, 9
  destroyed, 10
  game.h, 27
  game_over, 10
  game_over_dealy, 10
  lander, 10
  saved, 10
  successfull, 10
  time_started, 10
get_impact_force
  lander.c, 57
  lander.h, 31
get_lander_inertia
  lander.c, 58
  lander.h, 31
get_screen_coordinates
  camera.c, 50
  camera.h, 21
get_terrain_height
  terrain.c, 67
  terrain.h, 42
get_torque
  lander.c, 58
  lander.h, 32
get_world_coordinates
  camera.c, 50
  camera.h, 22

height
  Camera, 8
hover
  Button, 7
hover_color
  button.c, 49

impact_count
  Lander, 12
ImpactPoint, 11
  can_collide, 11
  lander.h, 30
  point, 11
include/button.h, 19, 20
include/camera.h, 20, 23
include/events.h, 24
include/file_handler.h, 25, 26
include/game.h, 26, 28
include/lander.h, 29, 34

include/leaderboard.h, 35, 36
include/main.h, 36, 37
include/menu.h, 37, 39
include/particle.h, 39, 41
include/terrain.h, 41, 43
include/text_io.h, 43, 44
include/vector.h, 44, 48
inertia_max
    lander.c, 61
inertia_min
    lander.c, 61
init_game
    game.c, 54
    game.h, 27
init_lander
    lander.c, 58
    lander.h, 32
init_leaderboard
    leaderboard.c, 62
    leaderboard.h, 35
init_menu
    menu.c, 64
    menu.h, 38
init_terrain
    terrain.c, 68
    terrain.h, 42
input_text
    text_io.c, 69
    text_io.h, 43

Lander, 11
    angular_velocity, 12
    dry_mass, 12
    engines, 12
    impact_count, 12
    lander.h, 30
    particle_system, 12
    position, 12
    propellant, 13
    rotation, 13
    velocity, 13
lander
    GameState, 10
lander.c
    bulk_add_particles, 56
    center_of_mass, 60
    dashboard_texture, 60
    destroy_lander, 57
    display_dashboard, 57
    dry_mass, 60
    friction_coefficient, 60
    g, 61
    get_impact_force, 57
    get_lander_inertia, 58
    get_torque, 58
    inertia_max, 61
    inertia_min, 61
    init_lander, 58
    lander_texture, 61
    lander_to_world_coord, 59
    lander_total_mass, 59
    main_engine_fuel_rate, 61
    main_engine_thrust, 61
    propellant_mass, 61
    rcs_fuel_rate, 61
    rcs_thrust, 62
    render_lander, 59
    size_px, 62
    to_metric, 60
    update_lander, 60
lander.h
    bulk_add_particles, 30
    destroy_lander, 31
    display_dashboard, 31
    Engine, 30
    get_impact_force, 31
    get_lander_inertia, 31
    get_torque, 32
    ImpactPoint, 30
    init_lander, 32
    Lander, 30
    lander_to_world_coord, 33
    lander_total_mass, 33
    LEFT_ENGINE, 30
    MAIN_ENGINE, 30
    render_lander, 33
    RIGHT_ENGINE, 30
    ROTATE_CCW, 30
    ROTATE_CW, 30
    update_lander, 33
lander_texture
    lander.c, 61
lander_to_world_coord
    lander.c, 59
    lander.h, 33
lander_total_mass
    lander.c, 59
    lander.h, 33
landing_quality
    game.c, 54
    game.h, 28
last
    List, 13
LEADERBOARD
    main.h, 37
    menu.h, 38
leaderboard.c
    count, 63
    destroy_leaderboard, 62
    font, 63
    init_leaderboard, 62
    render_leaderboard, 63
    score, 63
leaderboard.h
    destroy_leaderboard, 35
    init_leaderboard, 35
    render_leaderboard, 35

LEFT_ENGINE
    lander.h, 30
lerp
    camera.c, 50
    camera.h, 22
lerp2
    camera.c, 51
life_time
    Particle, 15
List, 13
    first, 13
    last, 13
    particle.h, 40
ListElement, 14
    next, 14
    particle, 14
    particle.h, 40
    prev, 14
lived
    Particle, 15

main
    main.c, 64
main.c
    main, 64
main.h
    GAME, 37
    LEADERBOARD, 37
    MENU, 37
    Screen, 36
MAIN_ENGINE
    lander.h, 30
main_engine_fuel_rate
    lander.c, 61
main_engine_thrust
    lander.c, 61
margin
    menu.c, 65
MENU
    main.h, 37
    menu.h, 38
menu.c
    button_count, 65
    buttons, 65
    destroy_menu, 64
    font, 65
    init_menu, 64
    margin, 65
    menu_events, 64
    render_menu, 65
menu.h
    destroy_menu, 38
    GAME, 38
    init_menu, 38
    LEADERBOARD, 38
    MENU, 38
    menu_events, 38
    render_menu, 38
    Screen, 37, 38
menu_events
    menu.c, 64
    menu.h, 38

name
    Score, 16
next
    ListElement, 14
noise
    terrain.c, 68

Particle, 15
    end_color, 15
    life_time, 15
    lived, 15
    particle.h, 40
    position, 15
    size, 15
    start_color, 15
    velocity, 16
particle
    ListElement, 14
particle.c
    append_particle, 66
    delete_particle, 66
    destroy_particles, 66
    render_particles, 66
    update_particles, 67
particle.h
    append_particle, 40
    delete_particle, 40
    destroy_particles, 40
    List, 40
    ListElement, 40
    Particle, 40
    render_particles, 40
    update_particles, 41
particle_system
    Lander, 12
PIXELS_PER_METER
    camera.c, 52
    camera.h, 23
point
    ImpactPoint, 11
position
    Camera, 8
    Lander, 12
    Particle, 15
prev
    ListElement, 14
propellant
    Lander, 13
propellant_mass
    lander.c, 61
pseudo_random
    terrain.c, 68

quality
    Score, 17

rcs_fuel_rate
  lander.c, 61
rcs_thrust
  lander.c, 62
read_scores
  file_handler.c, 52
  file_handler.h, 25
README.md, 48
rect
  Button, 7
render_button
  button.c, 49
  button.h, 20
render_game_over
  game.c, 54
  game.h, 28
render_lander
  lander.c, 59
  lander.h, 33
render_leaderboard
  leaderboard.c, 63
  leaderboard.h, 35
render_menu
  menu.c, 65
  menu.h, 38
render_particles
  particle.c, 66
  particle.h, 40
render_terrain
  terrain.c, 68
  terrain.h, 42
render_text_centered
  text_io.c, 69
  text_io.h, 44
renderer
  Camera, 8
RIGHT_ENGINE
  lander.h, 30
ROTATE_CCW
  lander.h, 30
ROTATE_CW
  lander.h, 30
rotation
  Lander, 13

save_state
  game.c, 54
  game.h, 28
saved
  GameState, 10
Score, 16
  file_handler.h, 25
  fuel, 16
  name, 16
  quality, 17
  score, 17
  time, 17
score
  leaderboard.c, 63
  Score, 17
Screen
  main.h, 36
  menu.h, 37, 38
size
  Particle, 15
size_px
  lander.c, 62
src/button.c, 48
src/camera.c, 49
src/file_handler.c, 52
src/game.c, 53
src/lander.c, 55
src/leaderboard.c, 62
src/main.c, 63
src/menu.c, 64
src/particle.c, 66
src/terrain.c, 67
src/text_io.c, 69
src/vector.c, 70
start_color
  Particle, 15
SUCCESS_EVENT_CODE
  events.h, 24
successfull
  GameState, 10

terrain.c
  get_terrain_height, 67
  init_terrain, 68
  noise, 68
  pseudo_random, 68
  render_terrain, 68
  terrain_max_height, 68
  TERRAIN_SEED, 69
terrain.h
  get_terrain_height, 42
  init_terrain, 42
  render_terrain, 42
  TERRAIN_SEED, 43
terrain_max_height
  terrain.c, 68
TERRAIN_SEED
  terrain.c, 69
  terrain.h, 43
text
  Button, 7
text_io.c
  input_text, 69
  render_text_centered, 69
  valid, 70
text_io.h
  input_text, 43
  render_text_centered, 44
time
  Score, 17
time_started
  GameState, 10
to_metric

lander.c, 60

update_camera
camera.c, 51
camera.h, 22
update_game
game.c, 55
game.h, 28
update_lander
lander.c, 60
lander.h, 33
update_particles
particle.c, 67
particle.h, 41

V_add
vector.c, 71
vector.h, 45
V_cross_product
vector.c, 71
vector.h, 45
V_divide_const
vector.c, 71
vector.h, 46
V_len
vector.c, 71
vector.h, 46
V_multiply
vector.c, 71
vector.h, 46
V_multiply_const
vector.c, 71
vector.h, 46
V_normalize
vector.c, 72
vector.h, 46
V_rotate
vector.c, 72
vector.h, 47
V_subtract
vector.c, 72
vector.h, 47
V_to_point
vector.c, 72
vector.h, 47
valid
text_io.c, 70
vector.c
V_add, 71
V_cross_product, 71
V_divide_const, 71
V_len, 71
V_multiply, 71
V_multiply_const, 71
V_normalize, 72
V_rotate, 72
V_subtract, 72
V_to_point, 72
vector.h
V_add, 45
V_cross_product, 45
V_divide_const, 46
V_len, 46
V_multiply, 46
V_multiply_const, 46
V_normalize, 46
V_rotate, 47
V_subtract, 47
V_to_point, 47
Vector2, 45
Vector2, 17
vector.h, 45
x, 17
y, 17
velocity
Lander, 13
Particle, 16

width
Camera, 9

x
Vector2, 17

y
Vector2, 17

zoom
Camera, 9