Worms!

Generated by Doxygen 1.8.1.2

Sun May 11 2014 17:13:44

Contents

Chapter 1

Worms!

Worms! is an ncurses based worms game.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

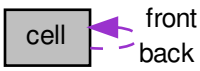
Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cell Struct Reference

Collaboration diagram for cell:



Data Fields

- int x
- int y
- struct cell * front
- struct cell * back

4.1.1 Detailed Description

Structure to hold an individual cell of the worm's body

4.1.2 Field Documentation

4.1.2.1 struct cell* cell::back

Pointer to next cell towards tail

4.1.2.2 struct cell* cell::front

Pointer to next cell towards head

4.1.2.3 int cell::x

X coordinate of the cell

4.1.2.4 int cell::y

Y coordinate of the cell

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

- worms_worm.c

4.2 game_window Struct Reference

Structure to hold details of game windows.

Data Fields

- WINDOW * window
- int rows
- int cols

4.2.1 Field Documentation

4.2.1.1 int game_window::cols

Number of columns in the window

4.2.1.2 int game_window::rows

Number of rows in the window

4.2.1.3 WINDOW* game_window::window

Pointer to curses WINDOW structure

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

- worms_screen.c

4.3 gw Struct Reference

Structure for main game window.

Data Fields

- WINDOW * window
- struct winsize ws
- int old_cursor
- bool initialized

4.3.1 Field Documentation

4.3.1.1 bool gw::initialized

true if initialized, false otherwise

4.3.1.2 int gw::old_cursor

To store the old cursor

4.3.1.3 WINDOW* gw::window

Pointer to main curses window

4.3.1.4 struct winsize gw::ws

Contains dimensions of terminal

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

- tge_curses_routines.c

4.4 tge_parameters Struct Reference

Structure for containing game parameters.

`#include <tge_main_game.h>`

Data Fields

- void(* setup_function)(void)
- void(* draw_function)(void)
- void(* input_function)(int)
- void(* teardown_function)(int)
- double timer_interval

4.4.1 Field Documentation

4.4.1.1 void(* tge_parameters::draw_function)(void)

Draw function

4.4.1.2 void(* tge_parameters::input_function)(int)

Input handling function

4.4.1.3 void(* tge_parameters::setup_function)(void)

Setup/initialization function

4.4.1.4 void(* tge_parameters::teardown_function)(int)

Cleanup function

4.4.1.5 double tge_parameters::timer_interval

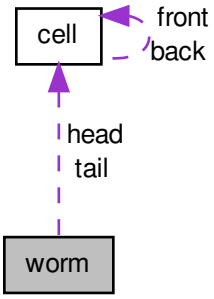
Timer interval, in seconds

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

- tge_main_game.h

4.5 worm Struct Reference

Collaboration diagram for worm:



Data Fields

- struct cell * head
- struct cell * tail
- enum worm_direction last_direction
- enum worm_direction next_direction

4.5.1 Detailed Description

Structure to hold the worm

4.5.2 Field Documentation

4.5.2.1 struct cell* worm::head

Pointer to head cell

4.5.2.2 enum worm_direction worm::last_direction

Next direction to move

4.5.2.3 enum worm_direction worm::next_direction

Last direction moved

4.5.2.4 struct cell* worm::tail

Pointer to tail cell

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

- worms_worm.c

Chapter 5

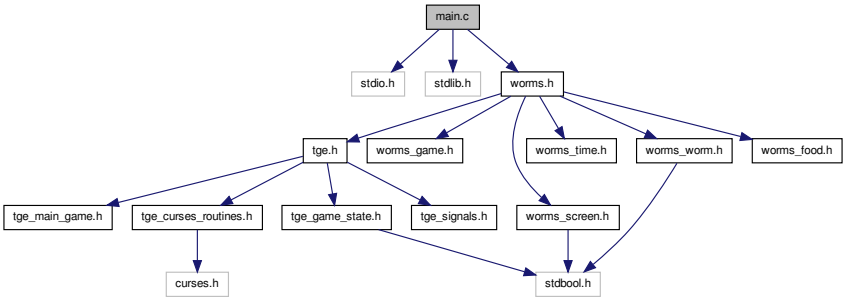
File Documentation

5.1 main.c File Reference

main() function for worms game.

```
#include <stdio.h>
#include <stdlib.h>
#include "worms.h"
```

Include dependency graph for main.c:



Functions

- void print_quit_message (const int end_status)

  Prints a quit message.
- int main (void)

  main() function.

5.1.1 Detailed Description

main() function for worms game.

Author

Paul Griffiths

Copyright

5.1.2 Function Documentation

5.1.2.1 int main (void)

main() function.

Returns

The exit status of the program.

5.1.2.2 void print_quit_message (const int *end_status*)

Prints a quit message.

Parameters

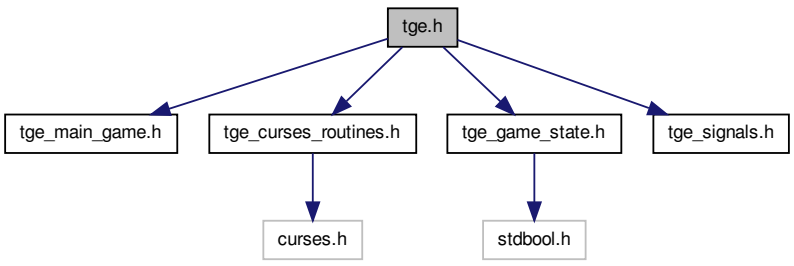
end_status	The exit status of the game.

5.2 tge.h File Reference

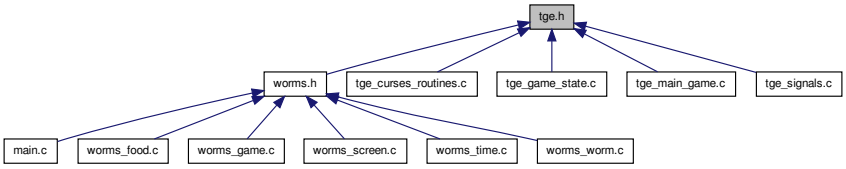
Aggregated header for curses timer game engine functions.

```
#include "tge_main_game.h"
#include "tge_curses_routines.h"
#include "tge_game_state.h"
#include "tge_signals.h"
```

Include dependency graph for tge.h:



This graph shows which files directly or indirectly include this file:



5.2.1 Detailed Description

Aggregated header for curses timer game engine functions.

A "timer game" is here defined as a game with two properties:

- the game and display regularly updates at a specified time interval; and
- the game can receive keyboard input at any time.

The rationale for providing this framework is twofold. Firstly, to encapsulate some of the overhead of setting up the game, including initializing curses, setting up signal handlers, and controlling the overall flow of the game. Secondly, to solve the programming problem of asynchronously updating the game while waiting on user input.

An initial naive implementation may be to implement the game timer using `SIGALRM`, and update the screen in the signal handler, while simply blocking on user input in the main thread. However, the curses functions are not re-entrant, and neither are many standard library functions, so this approach is not reliable. Similar objections would apply to a threads-based approach where one thread blocks on input, and a second thread updates the screen.

A second naive implementation would be to simply poll for input without blocking, which would be an unnecessary waste of processor time.

The solution here is to wait for input with `select()` to avoid blocking on any input. The game timer is implemented with `SIGALRM`, and the signal handler simply sets an "updated needed" variable and has the automatic side-effect of interrupting `select()` (note: portable curses programs cannot make any assumptions about whether handled signals will interrupt any curses input functions, so `select()` is necessary).

The library implements a game loop which begins by checking if a screen update is necessary. If it is, the screen is updated, and the loop re-entered. If it is not, the loop waits on `select()`. If no input is entered, `select()` will interrupt on handling `SIGALRM` and continue to the next iteration of the loop, and the screen will be updated again. If input is entered, `select()` will return and the input can be obtained without blocking.

Author

Paul Griffiths

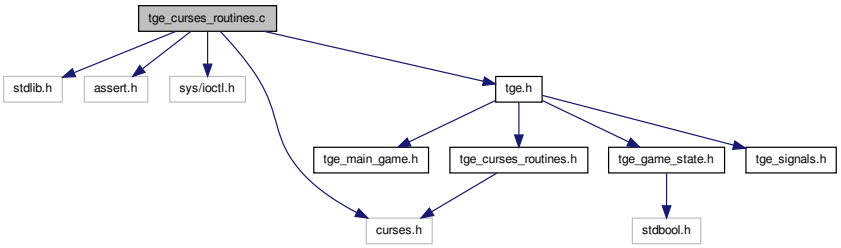
Copyright

5.3 tge_curses_routines.c File Reference

Implementation of TGE general curses functions.

```
#include <stdlib.h>
#include <assert.h>
#include <sys/ioctl.h>
#include <curses.h>
#include "tge.h"
```

Include dependency graph for tge_curses_routines.c:



Data Structures

- struct gw

  Structure for main game window.

Macros

- #define _POSIX_C_SOURCE 200809L

Functions

- void tge_initialize_screen (void)
- void tge_free_screen (void)
- WINDOW * tge_main_window (void)

  Returns a pointer to the main curses window.
- int tge_get_character (void)

  Gets a character input by the player.
- int tge_term_rows (void)

  Returns the number of rows in the terminal.
- int tge_term_cols (void)

  Returns the number of columns in the terminal.

Variables

- static struct gw game_window = {NULL, {0, 0, 0, 0}, 0, false}

5.3.1 Detailed Description

Implementation of TGE general curses functions.

Author

Paul Griffiths

Copyright

5.3.2 Macro Definition Documentation

5.3.2.1 #define _POSIX_C_SOURCE 200809L

POSIX feature test macro

5.3.3 Function Documentation

5.3.3.1 void tge_free_screen (void)

Frees and destroys the game screen.

5.3.3.2 int tge_get_character (void)

Gets a character input by the player. This function will not block if no input is ready.

Returns

The character input by the player, or -1 if no character was ready.

5.3.3.3 void tge_initialize_screen (void)

Initializes the game screen.

5.3.3.4 WINDOW* tge_main_window (void)

Returns a pointer to the main curses window.

Returns

A pointer to the main curses window.

5.3.3.5 int tge_term_cols (void)

Returns the number of columns in the terminal.

Returns

The number of columns in the terminal.

5.3.3.6 int tge_term_rows (void)

Returns the number of rows in the terminal.

Returns

The number of rows in the terminal.

5.3.4 Variable Documentation

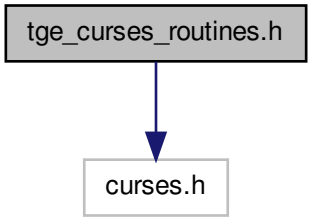
5.3.4.1 struct gw game_window = {NULL, {0, 0, 0, 0}, 0, false} [static]

File scope variable to hold main game window

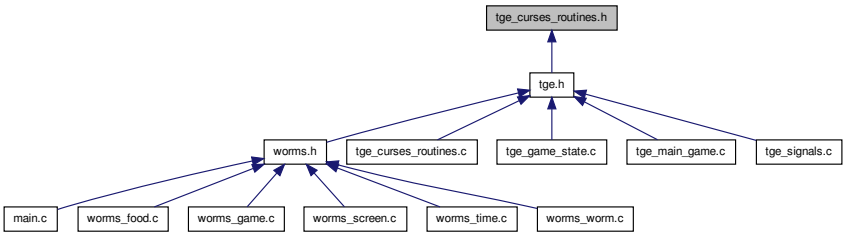
5.4 tge_curses_routines.h File Reference

Interface to TGE general curses functions.

`#include <curses.h>`
Include dependency graph for tge_curses_routines.h:



This graph shows which files directly or indirectly include this file:



Functions

- void tge_initialize_screen (void)
- void tge_free_screen (void)
- WINDOW * tge_main_window (void)

  Returns a pointer to the main curses window.
- int tge_get_character (void)

  Gets a character input by the player.
- int tge_term_rows (void)

  Returns the number of rows in the terminal.
- int tge_term_cols (void)

  Returns the number of columns in the terminal.

5.4.1 Detailed Description

Interface to TGE general curses functions.

Author

Paul Griffiths

Copyright

5.4.2 Function Documentation

5.4.2.1 void tge_free_screen (void)

Frees and destroys the game screen.

5.4.2.2 int tge_get_character (void)

Gets a character input by the player. This function will not block if no input is ready.

Returns

The character input by the player, or -1 if no character was ready.

5.4.2.3 void tge_initialize_screen (void)

Initializes the game screen.

5.4.2.4 WINDOW* tge_main_window (void)

Returns a pointer to the main curses window.

Returns

A pointer to the main curses window.

5.4.2.5 int tge_term_cols (void)

Returns the number of columns in the terminal.

Returns

The number of columns in the terminal.

5.4.2.6 int tge_term_rows (void)

Returns the number of rows in the terminal.

Returns

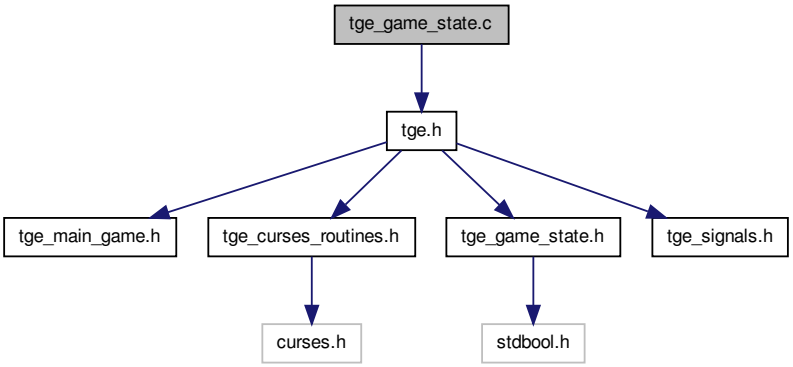
The number of rows in the terminal.

5.5 tge_game_state.c File Reference

Implementation of TGE game state functions.

```
#include "tge.h"
```

Include dependency graph for tge_game_state.c:



Enumerations

- enum game_state { TGE_GAME_STATE_NOTSTARTED, TGE_GAME_STATE_RUNNING, TGE_GAME_-STATE_ENDED }

  Enumeration constants for game state.

Functions

- void tge_start_game (void)
- bool tge_game_started (void)

  Tests if the timer game has started.
- void tge_end_game (const int status)

  Ends the timer game.
- bool tge_game_ended (void)

  Tests if the timer game has ended.
- int tge_end_status (void)

  Returns the timer game exit status.
- void tge_set_needs_refresh (const bool status)

  Notifies the library that the game needs updating.
- bool tge_needs_refresh (void)

  Checks if the game needs updating.

Variables

- static int tge_end_status_var = 0
- static enum game_state game_state = TGE_GAME_STATE_NOTSTARTED
- static bool refresh_flag = true

5.5.1 Detailed Description

Implementation of TGE game state functions.

Author

Paul Griffiths

Copyright

5.5.2 Enumeration Type Documentation

5.5.2.1 enum game_state

Enumerator:

TGE_GAME_STATE_NOTSTARTED Game has not started

TGE_GAME_STATE_RUNNING Game has started

TGE_GAME_STATE_ENDED Game has ended

5.5.3 Function Documentation

5.5.3.1 void tge_end_game (const int *status*)

Ends the timer game.

Parameters

status	The exit status of the game.

5.5.3.2 int tge_end_status (void)

Returns the timer game exit status. The meaning of this value is defined by the application using the TGE library.

Returns

The timer game exit status.

5.5.3.3 bool tge_game_ended (void)

Tests if the timer game has ended.

Returns

`true` if the timer game has ended, `false` otherwise.

5.5.3.4 bool tge_game_started (void)

Tests if the timer game has started.

Returns

`true` if the timer game has started, `false` otherwise.

5.5.3.5 bool tge_needs_refresh (void)

Checks if the game needs updating. This function returns the state set by `tge_set_needs_refresh()`.

Returns

`true` is the game needs updating, `false` otherwise.

5.5.3.6 void tge_set_needs_refresh (const bool *status*)

Notifies the library that the game needs updating. This is typically called by the `SIGALRM` signal handler in response to the game timer, but could be called by the application.

Parameters

status	`true` to turn on the "needs updating" flag, `false` to turn it off.

5.5.3.7 void tge_start_game (void)

Starts the timer game.

5.5.4 Variable Documentation

5.5.4.1 enum game_state game_state = TGE_GAME_STATE_NOTSTARTED [static]

File scope variable for current game state

5.5.4.2 bool refresh_flag = true [static]

File scope variable for "needs updating" flag

5.5.4.3 int tge_end_status_var = 0 [static]

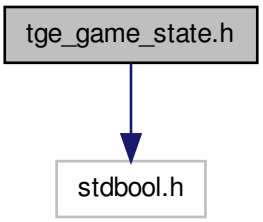
File scope variable for current exit status

5.6 tge_game_state.h File Reference

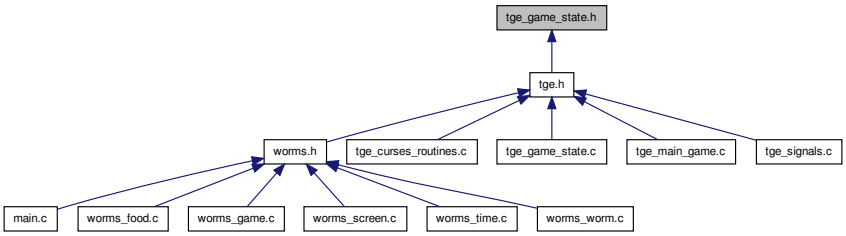
Interface to TGE game state functions.

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

Include dependency graph for tge_game_state.h:



This graph shows which files directly or indirectly include this file:



Functions

- void tge_start_game (void)
- bool tge_game_started (void)

  Tests if the timer game has started.
- void tge_end_game (const int status)

  Ends the timer game.
- bool tge_game_ended (void)

  Tests if the timer game has ended.
- int tge_end_status (void)

  Returns the timer game exit status.
- void tge_set_needs_refresh (const bool status)

  Notifies the library that the game needs updating.
- bool tge_needs_refresh (void)

  Checks if the game needs updating.

5.6.1 Detailed Description

Interface to TGE game state functions.

Author

Paul Griffiths

Copyright

5.6.2 Function Documentation

5.6.2.1 void tge_end_game (const int *status*)

Ends the timer game.

Parameters

status	The exit status of the game.

5.6.2.2 int tge_end_status (void)

Returns the timer game exit status. The meaning of this value is defined by the application using the TGE library.

Returns

The timer game exit status.

5.6.2.3 bool tge_game_ended (void)

Tests if the timer game has ended.

Returns

`true` if the timer game has ended, `false` otherwise.

5.6.2.4 bool tge_game_started (void)

Tests if the timer game has started.

Returns

`true` if the timer game has started, `false` otherwise.

5.6.2.5 bool tge_needs_refresh (void)

Checks if the game needs updating. This function returns the state set by `tge_set_needs_refresh()`.

Returns

`true` is the game needs updating, `false` otherwise.

5.6.2.6 void tge_set_needs_refresh (const bool *status*)

Notifies the library that the game needs updating. This is typically called by the `SIGALRM` signal handler in response to the game timer, but could be called by the application.

Parameters

status	`true` to turn on the "needs updating" flag, `false` to turn it off.

5.6.2.7 void tge_start_game (void)

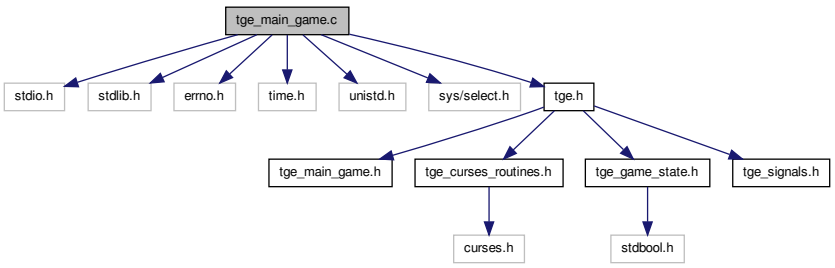
Starts the timer game.

5.7 tge_main_game.c File Reference

Implementation of TGE main game functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <sys/select.h>
#include "tge.h"
```

Include dependency graph for tge_main_game.c:



Macros

- #define _POSIX_C_SOURCE 200809L

Functions

- static void tge_game_loop (void)
- int tge_begin_game (const struct tge_parameters *parameters)

  Begins the timer game.

Variables

- static struct tge_parameters game_param

5.7.1 Detailed Description

Implementation of TGE main game functions.

Author

Paul Griffiths

Copyright

5.7.2 Macro Definition Documentation

5.7.2.1 #define _POSIX_C_SOURCE 200809L

POSIX feature test macro

5.7.3 Function Documentation

5.7.3.1 int tge_begin_game (const struct tge_parameters * *parameters*)

Begins the timer game.

Parameters

parameters	A pointer to a `struct tge_parameters` object containing the desired game parameters.

Returns

The exit status of the game.

5.7.3.2 static void tge_game_loop (void) [static]

Main game loop function.

5.7.4 Variable Documentation

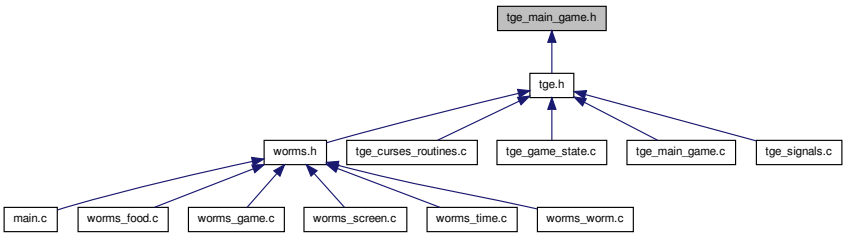
5.7.4.1 struct tge_parameters game_param [static]

File scope variable for game parameters

5.8 tge_main_game.h File Reference

Interface to curses timer game engine main game functions.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct tge_parameters

  Structure for containing game parameters.

Functions

- int tge_begin_game (const struct tge_parameters *parameters)

  Begins the timer game.
- void tge_end_game (const int status)

  Ends the timer game.

5.8.1 Detailed Description

Interface to curses timer game engine main game functions.

Author

Paul Griffiths

Copyright

5.8.2 Function Documentation

5.8.2.1 int tge_begin_game (const struct tge_parameters * *parameters*)

Begins the timer game.

Parameters

parameters	A pointer to a `struct tge_parameters` object containing the desired game parameters.

Returns

The exit status of the game.

5.8.2.2 void tge_end_game (const int *status*)

Ends the timer game.

Parameters

status	The exit status for the game.

Ends the timer game.

Parameters

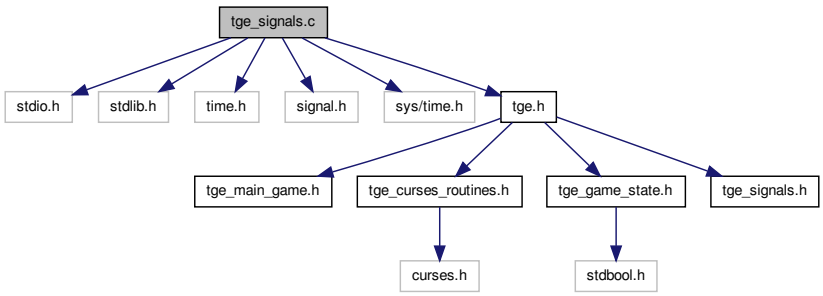
status	The exit status of the game.

5.9 tge_signals.c File Reference

Implementation of TGE signals functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <signal.h>
#include <sys/time.h>
#include "tge.h"
```

Include dependency graph for tge_signals.c:



Macros

- #define _POSIX_C_SOURCE 200809L

Functions

- static long tge_get_whole_seconds (const double seconds)

  Returns the whole seconds part of a seconds value.
- static long tge_get_fractional_microseconds (const double seconds)

  Returns the fractional microseconds part of a seconds value.
- static void tge_handler (int signum)

  Generic signal handler.
- void tge_set_signal_handlers (void)
- void tge_timer_start (const double start, const double interval)

Starts the game timer.

- void tge_timer_stop (void)

5.9.1 Detailed Description

Implementation of TGE signals functions.

Author

Paul Griffiths

Copyright

5.9.2 Macro Definition Documentation

5.9.2.1 #define _POSIX_C_SOURCE 200809L

POSIX feature test macro

5.9.3 Function Documentation

5.9.3.1 static long tge_get_fractional_microseconds (const double *seconds*) [static]

Returns the whole microseconds part of a seconds value.

Parameters

seconds	The seconds representation.

Returns

The fractional microseconds part of the seconds value.

5.9.3.2 static long tge_get_whole_seconds (const double *seconds*) [static]

Returns the whole seconds part of a seconds value.

Parameters

seconds	The seconds value.

Returns

The whole seconds part of the seconds value.

5.9.3.3 static void tge_handler (int *signum*) [static]

Generic signal handler.

Parameters

signum	The signal number.

5.9.3.4 void tge_set_signal_handlers (void)

Registers the signal handlers.

5.9.3.5 void tge_timer_start (const double *start*, const double *interval*)

Starts the game timer.

Parameters

start	The time until the first alarm, in seconds.
interval	The time interval between subsequent alarms, in seconds.

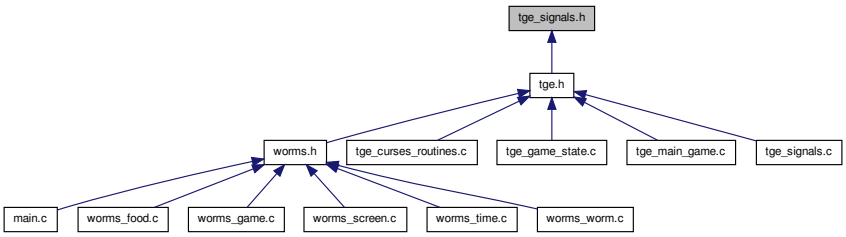
5.9.3.6 void tge_timer_stop (void)

Stops the game timer.

5.10 tge_signals.h File Reference

Interface to TGE signals functions.

This graph shows which files directly or indirectly include this file:



Functions

- void tge_set_signal_handlers (void)
- void tge_timer_start (const double start, const double interval)

  Starts the game timer.
- void tge_timer_stop (void)

5.10.1 Detailed Description

Interface to TGE signals functions.

Author

Paul Griffiths

Copyright

5.10.2 Function Documentation

5.10.2.1 void tge_set_signal_handlers (void)

Registers the signal handlers.

5.10.2.2 void tge_timer_start (const double *start*, const double *interval*)

Starts the game timer.

Parameters

start	The time until the first alarm, in seconds.
interval	The time interval between subsequent alarms, in seconds.

5.10.2.3 void tge_timer_stop (void)

Stops the game timer.

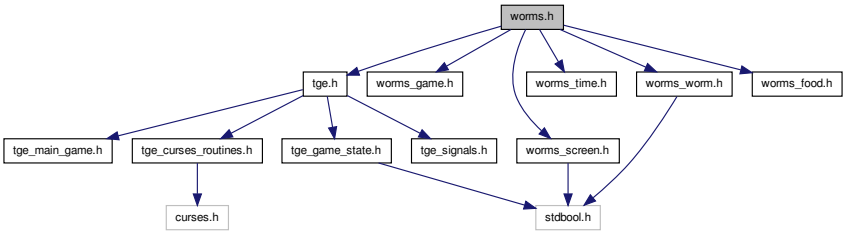
5.11 worms.dox File Reference

5.12 worms.h File Reference

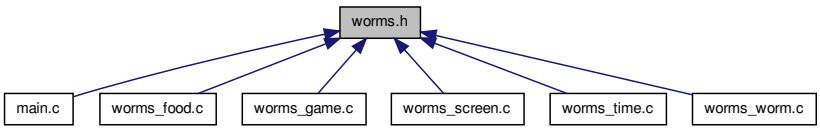
Aggregated header for worms game functions.

```
#include "tge.h"
#include "worms_game.h"
#include "worms_screen.h"
#include "worms_time.h"
#include "worms_worm.h"
#include "worms_food.h"
```

Include dependency graph for worms.h:



This graph shows which files directly or indirectly include this file:



5.12.1 Detailed Description

Aggregated header for worms game functions.

Author

Paul Griffiths

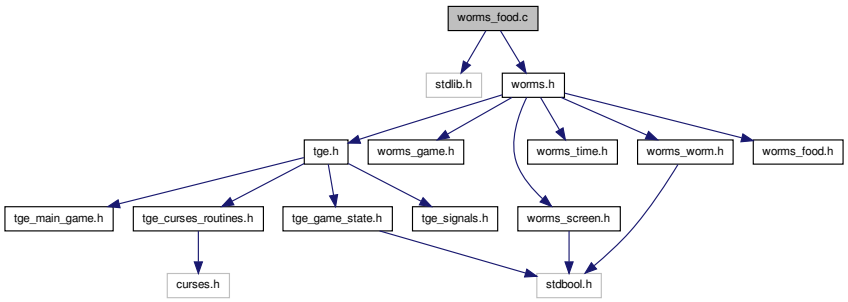
Copyright

5.13 worms_food.c File Reference

Implementation of worm food functions.

```
#include <stdlib.h>
#include "worms.h"
```

Include dependency graph for worms_food.c:



Functions

- void worms_place_new_food (void)
- void worms_draw_food (void)
- bool worms_food_here (const int x, const int y)

  Tests if a piece of food is at the specified coordinates.
- int worms_get_food_eaten (void)

  Returns the total number of pieces of food eaten.

Variables

- static int food_x = 0
- static int food_y = 0
- static int food_eaten = -1

5.13.1 Detailed Description

Implementation of worm food functions for curses worms game.

Author

Paul Griffiths

Copyright

5.13.2 Function Documentation

5.13.2.1 void worms_draw_food (void)

Draws the current piece of food.

5.13.2.2 bool worms_food_here (const int *x*, const int *y*)

Tests if a piece of food is at the specified coordinates.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

`true` if a piece of food is at the specified coordinates, `false` otherwise.

5.13.2.3 int worms_get_food_eaten (void)

Returns the total number of pieces of food eaten since the start of the game.

Returns

The total number of pieces of food eaten since the start of the game.

5.13.2.4 void worms_place_new_food (void)

Places a new piece of food at a random location.

5.13.3 Variable Documentation

5.13.3.1 int food_eaten = -1 [static]

File scope variable for total food eaten since start of game

5.13.3.2 int food_x = 0 [static]

File scope variable for x coordinate of current food piece

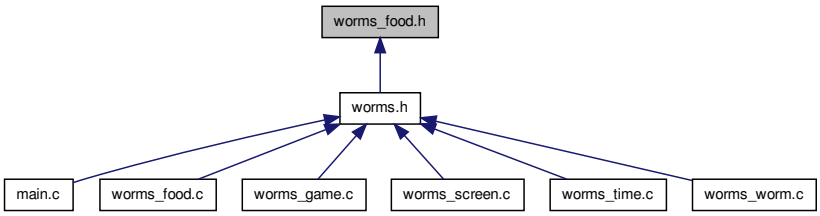
5.13.3.3 int food_y = 0 [static]

File scope variable for y coordinate of current food piece

5.14 worms_food.h File Reference

Interface to worm food functions.

This graph shows which files directly or indirectly include this file:



Functions

- void worms_place_new_food (void)
- void worms_draw_food (void)
- bool worms_food_here (const int x, const int y)

  Tests if a piece of food is at the specified coordinates.
- int worms_get_food_eaten (void)

  Returns the total number of pieces of food eaten.

5.14.1 Detailed Description

Interface to worm food functions for curses worms game.

Author

Paul Griffiths

Copyright

5.14.2 Function Documentation

5.14.2.1 void worms_draw_food (void)

Draws the current piece of food.

5.14.2.2 bool worms_food_here (const int *x*, const int *y*)

Tests if a piece of food is at the specified coordinates.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

`true` if a piece of food is at the specified coordinates, `false` otherwise.

5.14.2.3 int worms_get_food_eaten (void)

Returns the total number of pieces of food eaten since the start of the game.

Returns

The total number of pieces of food eaten since the start of the game.

5.14.2.4 void worms_place_new_food (void)

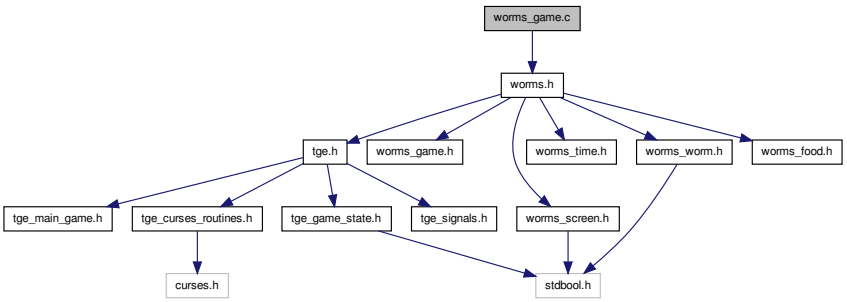
Places a new piece of food at a random location.

5.15 worms_game.c File Reference

Implementation of TGE game engine callback functions.

```
#include "worms.h"
```

Include dependency graph for worms_game.c:



Functions

- void worms_game_setup (void)

  Initializes the game.
- void worms_game_teardown (const int end_status)

  Cleans up after the game ends.
- void worms_draw_screen (void)
- void worms_process_input (const int ch)

  Processes keyboard input.

5.15.1 Detailed Description

Implementation of TGE game engine callback functions for curses worms game.

Author

Paul Griffiths

Copyright

5.15.2 Function Documentation

5.15.2.1 void worms_draw_screen (void)

Draws the game screen at each timer interval.

5.15.2.2 void worms_game_setup (void)

Initializes the game area, worm, worm food, and game time.

5.15.2.3 void worms_game_teardown (const int *end_status*)

Destroys the worm and the game area.

Parameters

end_status	Status code for end-of-game reason.

5.15.2.4 void worms_process_input (const int *ch*)

Processes keyboard input to move the worm or quit the game.

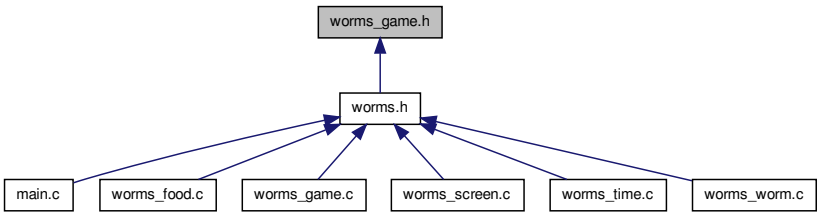
Parameters

ch	Character code representing the key pressed.

5.16 worms_game.h File Reference

Interface to TGE game engine callback functions.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum worms_exit_status { WORMS_EXIT_NORMAL, WORMS_EXIT_HITWALL, WORMS_EXIT_HITSELF }

Functions

- void worms_game_setup (void)

  Initializes the game.
- void worms_game_teardown (const int end_status)

  Cleans up after the game ends.
- void worms_draw_screen (void)
- void worms_process_input (const int ch)

  Processes keyboard input.

5.16.1 Detailed Description

Interface to TGE game engine callback functions, along with general game data structures and constants.

Author

Paul Griffiths

Copyright

5.16.2 Enumeration Type Documentation

5.16.2.1 enum worms_exit_status

Enumeration constants for game exit status.

Enumerator:

WORMS_EXIT_NORMAL Player chose to quit

WORMS_EXIT_HITWALL Worm ran into the arena wall

WORMS_EXIT_HITSELF Worm ran into its own body

5.16.3 Function Documentation

5.16.3.1 void worms_draw_screen (void)

Draws the game screen at each timer interval.

5.16.3.2 void worms_game_setup (void)

Initializes the game area, worm, worm food, and game time.

5.16.3.3 void worms_game_teardown (const int *end_status*)

Destroys the worm and the game area.

Parameters

end_status	Status code for end-of-game reason.

5.16.3.4 void worms_process_input (const int *ch*)

Processes keyboard input to move the worm or quit the game.

Parameters

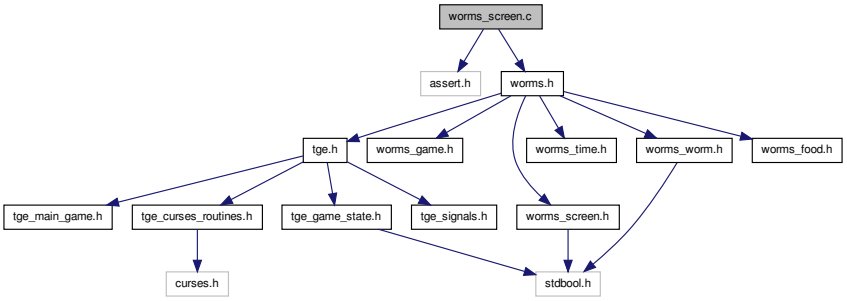
ch	Character code representing the key pressed.

5.17 worms_screen.c File Reference

Implementation of worms game screen functions.

```
#include <assert.h>
#include "worms.h"
```

Include dependency graph for worms_screen.c:



Data Structures

- struct game_window

  Structure to hold details of game windows.

Functions

- static void worms_draw_title_window (void)
- static void worms_draw_info_window (void)
- void worms_game_area_init (void)
- void worms_game_area_destroy (void)
- void worms_draw_arena_border (void)
- void worms_draw_sidebar (void)
- void worms_refresh_game_area (void)
- int worms_game_arena_cols (void)

  Returns the number of columns in the game arena.
- int worms_game_arena_rows (void)

  Returns the number of rows in the game arena.
- void worms_write_arena_character (const enum worms_cell_character ch, const int x, const int y)

  Writes a character to the game arena.
- enum worms_cell_character worms_get_arena_character (const int x, const int y)

  Returns the character at the specified arena coordinates.
- bool worms_coords_in_game_arena (const int x, const int y)

  Tests if the specified coordinates are within the arena.

Variables

- static const int sidebar_cols = 20
- static const int title_rows = 7
- static struct game_window arena_window
- static struct game_window title_window
- static struct game_window info_window

5.17.1 Detailed Description

Implementation of worms game screen functions. The game screen consists of two parts: the "arena", where the worm moves, and the "sidebar", containing the game title and game statistics. "Game area" can also refer to the entire game screen.

Author

Paul Griffiths

Copyright

5.17.2 Function Documentation

5.17.2.1 bool worms_coords_in_game_arena (const int *x*, const int *y*)

Tests if the specified coordinates are within the arena. The coordinates are *not* considered to be within the arena if they are within the arena's border, so this test is suitable for determining whether the worm has hit the border at the edge of the arena.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

`true` if the specified coordinates are within the arena, excluding the arena border, and `false` otherwise.

5.17.2.2 void worms_draw_arena_border (void)

Draws a border around the game arena.

5.17.2.3 static void worms_draw_info_window (void) [static]

Draws the information window in the sidebar.

5.17.2.4 void worms_draw_sidebar (void)

Draws the sidebar.

5.17.2.5 static void worms_draw_title_window (void) [static]

Draws the title window in the sidebar.

5.17.2.6 void worms_game_area_destroy (void)

Destroys the game area, including the sidebar.

5.17.2.7 void worms_game_area_init (void)

Initializes the game area, including the sidebar.

5.17.2.8 int worms_game_arena_cols (void)

Returns the number of columns in the game arena. The number of columns *includes* the arena border, which is one character wide on all sides.

Returns

The number of columns in the game arena.

5.17.2.9 int worms_game_arena_rows (void)

Returns the number of rows in the game arena. The number of rows *includes* the arena border, which is one character wide on all sides.

Returns

The number of rows in the game arena.

5.17.2.10 enum worms_cell_character worms_get_arena_character (const int *x*, const int *y*)

Returns the character at the specified arena coordinates.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

The character at the specified coordinates in the arena.

5.17.2.11 void worms_refresh_game_area (void)

Refreshes the main arena and the sidebar.

5.17.2.12 void worms_write_arena_character (const enum worms_cell_character *ch*, const int *x*, const int *y*)

Writes a character to the game arena.

Parameters

ch	The character to write.
x	The x coordinate at which to write the character.
y	The y coordinate at which to write the character.

5.17.3 Variable Documentation

5.17.3.1 struct game_window arena_window [static]

File scope variable for main arena window

5.17.3.2 struct game_window info_window [static]

File scope variable for information window

5.17.3.3 const int sidebar_cols = 20 [static]

File scope variable for number of columns in sidebar

5.17.3.4 const int title_rows = 7 [static]

File scope variable for number of rows in title window

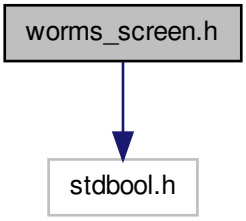
5.17.3.5 struct game_window title_window [static]

File scope variable for title window

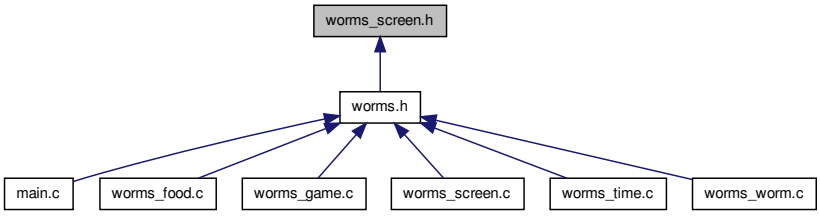
5.18 worms_screen.h File Reference

Interface to worms game screen functions.

`#include <stdbool.h>`
Include dependency graph for worms_screen.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum worms_cell_character { WORM_BODY_CHARACTER, WORM_EMPTY_CHARACTER, WORM_FOOD_CHARACTER }

Functions

- void worms_game_area_init (void)
- void worms_game_area_destroy (void)

- void worms_draw_arena_border (void)
- void worms_draw_sidebar (void)
- void worms_refresh_game_area (void)
- int worms_game_arena_cols (void)

  Returns the number of columns in the game arena.
- int worms_game_arena_rows (void)

  Returns the number of rows in the game arena.
- void worms_write_arena_character (const enum worms_cell_character ch, const int x, const int y)

  Writes a character to the game arena.
- enum worms_cell_character worms_get_arena_character (const int x, const int y)

  Returns the character at the specified arena coordinates.
- bool worms_coords_in_game_arena (const int x, const int y)

  Tests if the specified coordinates are within the arena.

5.18.1 Detailed Description

Interface to worms game screen functions. The game screen consists of two parts: the "arena", where the worm moves, and the "sidebar", containing the game title and game statistics. "Game area" can also refer to the entire game screen.

Author

Paul Griffiths

Copyright

5.18.2 Enumeration Type Documentation

5.18.2.1 enum worms_cell_character

Enumeration constants for game cell characters

Enumerator:

WORM_BODY_CHARACTER Character for worm body

WORM_EMPTY_CHARACTER Character for empty cell

WORM_FOOD_CHARACTER Character for piece of food

5.18.3 Function Documentation

5.18.3.1 bool worms_coords_in_game_arena (const int *x*, const int *y*)

Tests if the specified coordinates are within the arena. The coordinates are *not* considered to be within the arena if they are within the arena's border, so this test is suitable for determining whether the worm has hit the border at the edge of the arena.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

`true` if the specified coordinates are within the arena, excluding the arena border, and `false` otherwise.

5.18.3.2 void worms_draw_arena_border (void)

Draws a border around the game arena.

5.18.3.3 void worms_draw_sidebar (void)

Draws the sidebar.

5.18.3.4 void worms_game_area_destroy (void)

Destroys the game area, including the sidebar.

5.18.3.5 void worms_game_area_init (void)

Initializes the game area, including the sidebar.

5.18.3.6 int worms_game_arena_cols (void)

Returns the number of columns in the game arena. The number of columns *includes* the arena border, which is one character wide on all sides.

Returns

The number of columns in the game arena.

5.18.3.7 int worms_game_arena_rows (void)

Returns the number of rows in the game arena. The number of rows *includes* the arena border, which is one character wide on all sides.

Returns

The number of rows in the game arena.

5.18.3.8 enum worms_cell_character worms_get_arena_character (const int *x*, const int *y*)

Returns the character at the specified arena coordinates.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

The character at the specified coordinates in the arena.

5.18.3.9 void worms_refresh_game_area (void)

Refreshes the main arena and the sidebar.

5.18.3.10 void worms_write_arena_character (const enum worms_cell_character *ch*, const int *x*, const int *y*)

Writes a character to the game arena.

Parameters

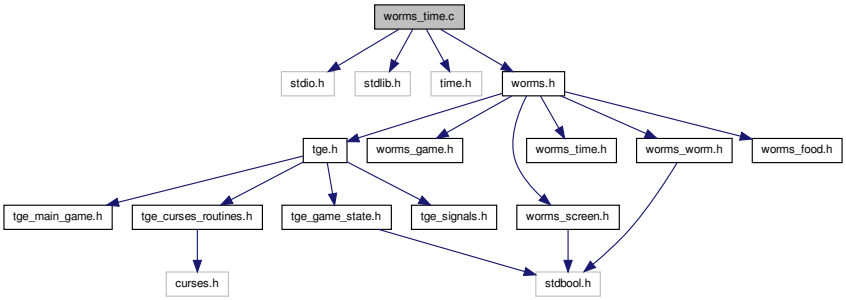
ch	The character to write.
x	The x coordinate at which to write the character.
y	The y coordinate at which to write the character.

5.19 worms_time.c File Reference

Implementation of worms game duration timer functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "worms.h"
```

Include dependency graph for worms_time.c:



Functions

- void worms_time_init (void)
- long worms_game_time (void)

  Returns the number of seconds since the game started.
- char * worms_game_time_string (const bool long_format)

  Returns a string representation of total game time.

Variables

- static time_t start_time

5.19.1 Detailed Description

Implementation of worms game duration timer functions.

Author

Paul Griffiths

Copyright

5.19.2 Function Documentation

5.19.2.1 long worms_game_time (void)

Returns the number of seconds since the game started.

Returns

The number of seconds since the game started.

5.19.2.2 char* worms_game_time_string (const bool *long_format*)

Returns a string representation of total game time.

Parameters

long_format	If this parameter is `true`, the string representation is of the form "[H] hours, [M] minutes, and [S] seconds". If `false`, the string representation is of the form "HH:MM:SS".

Returns

A string representation of the total game time. The returned pointer points to statically allocated storage, and is overwritten each time this function is called.

5.19.2.3 void worms_time_init (void)

Initializes the game duration timer.

5.19.3 Variable Documentation

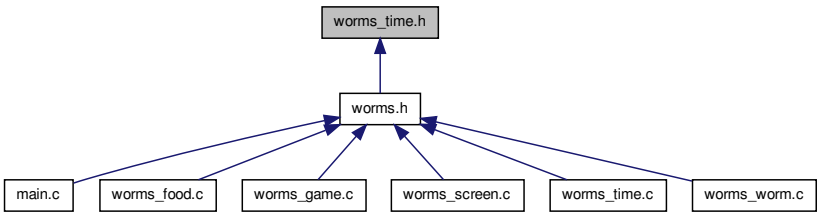
5.19.3.1 time_t start_time [static]

File scope variable to hold the time at the start of the game

5.20 worms_time.h File Reference

Interface to worms game duration timer functions.

This graph shows which files directly or indirectly include this file:



Functions

- void worms_time_init (void)
- long worms_game_time (void)

  Returns the number of seconds since the game started.
- char * worms_game_time_string (const bool long_format)

  Returns a string representation of total game time.

5.20.1 Detailed Description

Interface to worms game duration timer functions.

Author

Paul Griffiths

Copyright

5.20.2 Function Documentation

5.20.2.1 long worms_game_time (void)

Returns the number of seconds since the game started.

Returns

The number of seconds since the game started.

5.20.2.2 char* worms_game_time_string (const bool *long_format*)

Returns a string representation of total game time.

Parameters

long_format	If this parameter is `true`, the string representation is of the form "[H] hours, [M] minutes, and [S] seconds". If `false`, the string representation is of the form "HH:MM:SS".

Returns

A string representation of the total game time. The returned pointer points to statically allocated storage, and is overwritten each time this function is called.

5.20.2.3 void worms_time_init (void)

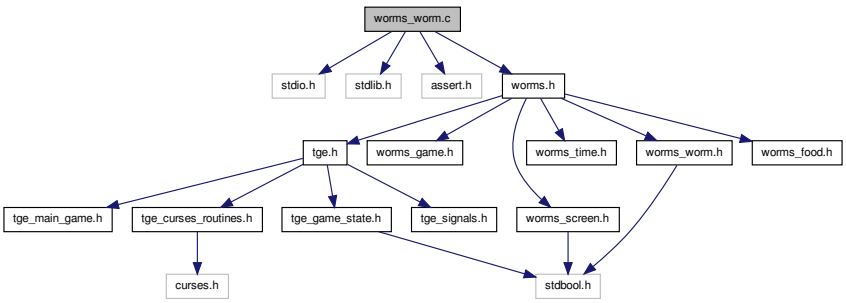
Initializes the game duration timer.

5.21 worms_worm.c File Reference

Implementation of worms game worm functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "worms.h"
```

Include dependency graph for worms_worm.c:



Data Structures

- struct cell
- struct worm

Functions

- static void worm_clear (void)

  Erases the worm from the arena.
- static bool worm_move (void)

  Moves the worm.
- static void worm_draw (void)
- static bool worm_cell_here (const int x, const int y)

  Tests if the specified coordinates contain a worm body cell.
- static void worm_add_cell_head (const int x, const int y)

  Adds a cell at the head of the worm.
- static void worm_delete_cell_head (void)
- static void worm_delete_cell_tail (void)
- void worm_init (void)

  Initializes the worm.

- void worm_destroy (void)

  Destroys the worm.
- void worm_set_direction (enum worm_direction direction)

  Attempts to change the next direction of the worm.
- bool worm_move_and_draw (void)

  Moves and draws the worm.

Variables

- static struct worm worm = {NULL, NULL, WORM_DIR_RIGHT, WORM_DIR_RIGHT}
- static const size_t WORM_START_LENGTH = 8

5.21.1 Detailed Description

Implementation of worms game worm functions. The worm is implemented as a double-ended, doubly-linked list.

Author

Paul Griffiths

Copyright

5.21.2 Function Documentation

5.21.2.1 static void worm_add_cell_head (const int *x*, const int *y*) [static]

Adds a cell at the head of the worm.

Parameters

x	The x coordinate of the new head.
y	The y coordinate of the new head.

5.21.2.2 static bool worm_cell_here (const int *x*, const int *y*) [static]

Tests if the specified coordinates contain a worm body cell. This function is used to test for collision with the worm's own body.

Parameters

x	The specified x coordinate.
y	The specified y coordinate.

Returns

`true` if the specified coordinates contain a worm body cell, `false` otherwise.

5.21.2.3 static void worm_clear (void) [static]

Erases the worm from the arena, i.e. overwrites it with empty cell characters.

5.21.2.4 static void worm_delete_cell_head (void) [static]

Deletes the cell at the head of the worm.

5.21.2.5 static void worm_delete_cell_tail (void) [static]

Deletes the cell at the tail of the worm.

5.21.2.6 void worm_destroy (void)

Destroys and deallocates memory for the worm.

5.21.2.7 static void worm_draw (void) [static]

Draws the worm.

5.21.2.8 void worm_init (void)

Initializes and allocates memory for the worm.

5.21.2.9 static bool worm_move (void) [static]

Moves the worm. The worm moves in the next direction specified in the worm structure. The function detects if the move would cause the worm to eat some food, and whether it would cause the worm to hit the arena wall, or its own body. In the former case, the tail cell is not deleted, and the worm grows in length. In the latter case, the game is ended.

Returns

`true` if the move caused the worm to eat some food, `false` in all other situations.

5.21.2.10 bool worm_move_and_draw (void)

Moves and draws the worm.

Returns

`true` if the movement that was performed caused the worm to eat a piece of food, `false` in all other situations.

5.21.2.11 void worm_set_direction (enum worm_direction *direction*)

Attempts to change the next direction of the worm. The worm will not actually change direction until the game next updates, and a second call to this function in the interim will nullify the effect of any previous calls, so it is safe to call this function an arbitrary number of times in between game updates. This function will ignore any attempt to change the direction in a way which is not allowed, particularly that the worm is not allowed to turn 180 degrees as this would cause it to hit its own body, so it is safe to call this function with any direction, and to trust it to ignore any direction changes which are not allowed.

Parameters

direction	The desired direction of the worm on the next update.

5.21.3 Variable Documentation

5.21.3.1 struct worm worm = {NULL, NULL, WORM_DIR_RIGHT, WORM_DIR_RIGHT} [static]

File scope variable to contain the worm

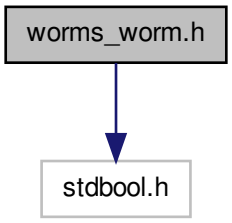
5.21.3.2 const size_t WORM_START_LENGTH = 8 [static]

File scope constant containing initial length of worm

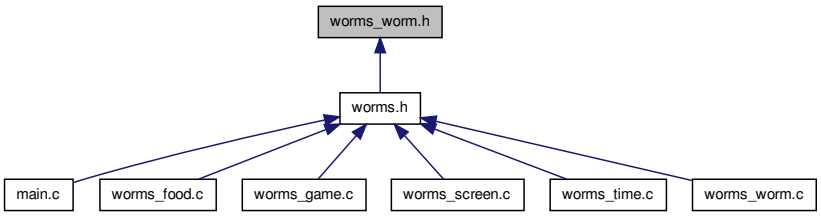
5.22 worms_worm.h File Reference

Interface to worms game worm functions.

`#include <stdbool.h>`
Include dependency graph for worms_worm.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum worm_direction { WORM_DIR_UP, WORM_DIR_RIGHT, WORM_DIR_DOWN, WORM_DIR_LEFT }
  Enumeration constants for worm movement direction.

Functions

- void worm_init (void)

Initializes the worm.

- void worm_destroy (void)

  Destroys the worm.

- void worm_set_direction (enum worm_direction direction)

  Attempts to change the next direction of the worm.

- bool worm_move_and_draw (void)

  Moves and draws the worm.

5.22.1 Detailed Description

Interface to worms game worm functions.

Author

Paul Griffiths

Copyright

5.22.2 Enumeration Type Documentation

5.22.2.1 enum worm_direction

Enumeration constants for worm movement direction. The order in which these are defined is meaningful to the implementation, namely that pairs of opposing directions (i.e. up versus down, left versus right) have values with an absolute difference of 2.

Enumerator:

WORM_DIR_UP Up direction

WORM_DIR_RIGHT Right direction

WORM_DIR_DOWN Down direction

WORM_DIR_LEFT Left direction

5.22.3 Function Documentation

5.22.3.1 void worm_destroy (void)

Destroys and deallocates memory for the worm.

5.22.3.2 void worm_init (void)

Initializes and allocates memory for the worm.

5.22.3.3 bool worm_move_and_draw (void)

Moves and draws the worm.

Returns

`true` if the movement that was performed caused the worm to eat a piece of food, `false` in all other situations.

5.22.3.4 void worm_set_direction (enum worm_direction *direction*)

Attempts to change the next direction of the worm. The worm will not actually change direction until the game next updates, and a second call to this function in the interim will nullify the effect of any previous calls, so it is safe to call this function an arbitrary number of times in between game updates. This function will ignore any attempt to change the direction in a way which is not allowed, particularly that the worm is not allowed to turn 180 degrees as this would cause it to hit its own body, so it is safe to call this function with any direction, and to trust it to ignore any direction changes which are not allowed.

Parameters

direction	The desired direction of the worm on the next update.

Index