Drawsystem

Generated by Doxygen 1.8.9.1

Sun May 10 2015 10:49:25

Contents

Chapter 1

Data Structure Index

1.1   Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 btn Struct Reference

```
#include <structs.h>
```

Data Fields

- coordinate tl_corner

  Coordinate of the top left corner of the box.
- int width

  Width of the button.
- int height

  Height of the button.
- TTF_Font ∗ font

  A font to be used for the text lable on the button.
- char text [40]

  A characer string to hold the buttons lable.
- SDL_Colour colour

  The colour of the button.

3.1.1 Detailed Description

A structure for holding all the information required for creating a button on the screen.

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

- src/structs.h

3.2 coordinate Struct Reference

```
#include <structs.h>
```

Data Fields

- double x_pos

  x positon of the coordinate pair.
- double y_pos

  y position of the coordinate pair.

3.2.1 Detailed Description

A coordinate structure to hold an (x, y) coordinate pair.

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

- src/structs.h

3.3 line Struct Reference

`#include <structs.h>`

Data Fields

- coordinate start

    Coordinate sructure for the start of the line.
- coordinate end

    Coordinate structure for the end of the line.

3.3.1 Detailed Description

A line structure that holds a coordinate pair that defines a line between two points

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

- src/structs.h

3.4 lsystem Struct Reference

`#include <structs.h>`

Data Fields

- char name [40]

    A character string for the name of the L-System.
- char axiom [40]

    A haracter array for the starting string (axiom) of the L-System.
- SDL_Colour bg_colour

    Background colour for the drawing screen.
- SDL_Colour ln_colour

    Line colour for the drawing screen.
- coordinate start

    Starting point for the turtle drawing the L-System.
- double angle

    Angle for the turtle to turn through each time the angle is increased or decreaseed.
- int length

    Length of each individual line to be drawn.
- int iterations

    Fractal depth of the L-System (how many times the character replacement is performed).
- int iteration_limit

> *A maximum fractal depth which can be different for different L-Systems so that the program is not drawing too much and stays fast.*

- int img_file_num

  A counter that counts how many images have been saved so that conflicting names are not produced in a single run of the program.

- int seq_file_num

  A counter that counts how many images have been saved to a sequence so that conflicting names are not produced in a single run of the program.

- char ∗ string

  A pointer to the L-System string to be drawn.

- line ∗ line_list

  A opinter to an array of lines that are calculated using the string.

- int line_list_length

  A counter that recrds the length of the line list.

- int remake_lines_flag

  A flag which tells the program to recalculate the line list.

- int remake_string_flag

  A flag which tells the program to recalculate the string.

- int info_disp_flag

  A flag which tells the program to display the L-System info to the screen.

- char rule_A [40]

  character replacement string for the 'A' chracter.

- char rule_B [40]

  character replacement string for the 'B' chracter.

- char rule_F [40]

  character replacement string for the 'F' chracter.

- char rule_f [40]

  character replacement string for the 'f' chracter.

- char rule_X [40]

  character replacement string for the 'X' chracter.

- char rule_Y [40]

  character replacement string for the 'Y' chracter.

- char rule_plus [40]

  character replacement string for the '+' chracter.

- char rule_minus [40]

  character replacement string for the '-' chracter.

- char rule_store [40]

  character replacement string for the '[' chracter.

- char rule_pop [40]

  character replacement string for the ']' chracter.

3.4.1 Detailed Description

A structure that holds all of the information required for the creation and drawing of an lsystem

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

- src/structs.h

3.5 turtle_state Struct Reference

```
#include <structs.h>
```

Data Fields

- coordinate pos

    Current position of the turtle.
- double heading

    Heading of the turtle (stored as radians).
- struct turtle_state ∗ next

    Pointer to the next element in a linked list.

3.5.1 Detailed Description

A structure that stores information on a turtle, including a pointer to the next turtle for the creation of a linked list that will be used to create a turtle stack.

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

- src/structs.h

Chapter 4

File Documentation

4.1 src/lsys.c File Reference

A source file for functions used in the creation of lsystem strings.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "structs.h"
#include "lsys.h"
```

Macros

- #define M_PI 3.14156265359

    A macro definition for the mathematical constant Pi.

Functions

- void iteration (lsystem ∗lsys)
- int makeString (lsystem ∗lsys)
- void initLsystem (lsystem ∗lsys)
- double dtor (double degrees)
- double rtod (double radians)
- void resetString (lsystem ∗lsys)
- void resetLines (lsystem ∗lsys)
- void sierpinski (lsystem ∗lsys)
- void dragon (lsystem ∗lsys)
- void plant1 (lsystem ∗lsys)
- void plant2 (lsystem ∗lsys)
- void islands (lsystem ∗lsys)
- void snowflake (lsystem ∗lsys)
- void quadKoch (lsystem ∗lsys)
- void gosper (lsystem ∗lsys)

4.1.1 Detailed Description

A source file for functions used in the creation of lsystem strings.

The L-System string is created by performing a character replacement on a starting string (axiom) using a set of character replacement rules and then recursivly calling the same character replacement reules on the result of the previous string to the required depth of iterations.

4.1.2 Function Documentation

4.1.2.1 void iteration (lsystem ∗ *lsys*)

Performes a single iteration of character replacement for a string.

This is done by firstly counting up the new length of the string, then allocating the required amout of memory before finaly creating the new string, freeing the old one, thensetting the poonter to point at the newly created string.

Parameters

out	*lsys*	the lsystem that holds the rules for character replacement and the string

4.1.2.2 int makeString (lsystem ∗ *lsys*)

takes in the axiom and the rules and produces the final string, using the number of iterations specified by the user.

Firstly, memory is allocated for the axiom to be places in the string container, this allocation is then checked before the iteration() function is then run on the string for the required number of iterations.

Parameters

out	*lsys*	a pointer to the lsystem that contaisn the rules, axiom, old string, and number of iterations.

Returns

returns 1 if sucessfull, and 0 if memory allocation failed.

4.1.2.3 void initLsystem (lsystem ∗ *lsys*)

Sets all rules in an lsystem to map to them selves so that there will be a default rule for each character in the string. Also intiialises other values that will be the default values for creation/drawing ect.

Parameters

out	*lsys*	an lsystem that will store the information.

4.1.2.4 double dtor (double *degrees*)

converts an angle from degrees into radians.

[in] degrees angle to be converted into radians.

Returns

angle in radians.

4.1.2.5 double rtod (double *radians*)

Converts an angle from radians into degrees.

Parameters

in	*radians*	angle to be converted into degrees.

Returns

angle in dgrees

4.1.2.6 void resetString (lsystem ∗ *lsys*)

Frees the pointers string and resets the flag.

Parameters

out	*lsys*	lsystem that holds the data to be freed.

4.1.2.7 void resetLines (lsystem ∗ *lsys*)

Frees the line list and resets the flag.

Parameters

out	*lsys*	lsystem that holds data to be freed

4.1.2.8 void sierpinski (lsystem ∗ *lsys*)

Copy Sierpinski Triangle rules to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to

4.1.2.9 void dragon (lsystem ∗ *lsys*)

Copy Dragon Curve rules to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to

4.1.2.10 void plant1 (lsystem ∗ *lsys*)

Copy Fractal Plant 1 rules to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to

4.1.2.11 void plant2 (lsystem ∗ *lsys*)

Copy the rules for a fractal plant to the lsystem.

Parameters

out	*lsys*	system for the rules to be copied to.

4.1.2.12 void islands (lsystem ∗ *lsys*)

Copy island and lake rules to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to.

4.1.2.13 void snowflake (lsystem ∗ *lsys*)

Copy Koch Snowflake rules to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to

4.1.2.14 void quadKoch (lsystem ∗ *lsys*)

Copy lsystem rules for a quadratic koch curve to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied too.

4.1.2.15 void gosper (lsystem ∗ *lsys*)

Copy lsystem rules for a gosper curve to the lsystem.

Parameters

out	*lsys*	lsystem for the rules to be copied to.

4.2 src/main.c File Reference

A source file for the main finction that will run the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "structs.h"
#include "lsys.h"
#include "ui.h"
```

Functions

- int init ()
- int main ()

4.2.1 Detailed Description

A source file for the main finction that will run the program.

Firslty all SDL initilaisation functions are run and checked for sucess. Then variables for the L-System and rendere-ing functions are initilaised before being passed to the relevant functions for default values to be set up.

Once all variables have been declared and initilaised, a while loop that checks for events and handles window drawing operations for different click events is entered and only exited when the program is quit.

When the program is quit, all variables that need freeing/destroying are handlend and the program exits.

4.2.2 Function Documentation

4.2.2.1 int init ( )

Initilaising SDL elements and printing errors if initialisation failed.

Returns

: true if errors occured during initilaisation.

4.2.2.2 int main ( )

Intialisinf variables for, and controling the running of the program.

Initialises all varaibles with default values and defines the button sets for each window before frawing the rewuired window defined by the win_flag. Runs functions for handeling click events for changing the win_flag and parameters within the L-System.

Returns

0 if the program runs sucessfully.

4.3 src/structs.c File Reference

A source file for functions used ito intitialise structures requied for the program.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "structs.h"
```

Functions

- void structInitLsystem (lsystem *lsys)
- void structInitCoord (coordinate *coord)
- void structInitLine (line *ln)
- void structInitBtn (btn *ui_button)
- void structInitTurtleState (turtle_state *turtle)

4.3.1 Detailed Description

A source file for functions used ito intitialise structures requied for the program.

Initialisation functions are required to insure defined behavior for every element in each structure.

4.3.2 Function Documentation

4.3.2.1 void structInitLsystem (lsystem ∗ *lsys*)

Initilaises an lsystem structure.

For use when declaring an lsystem to ensure that all elements have defined values and therfor predictable behavior.

Parameters

out	*lsys*	The lsystem structure to be initialized.

4.3.2.2 void structInitCoord (coordinate ∗ *coord*)

Initilaises a coordinate structure.

For use when declaring a coordinate to ensure that all elements have defined values and therfor predictable behavior.

Parameters

out	*coord*	The coordinate structue to be initialised.

4.3.2.3 void structInitLine (line ∗ *ln*)

Nnitilaises a line structure.

For use when declaring a line to ensure that all elements hae defined values and predictable behavior.

Parameters

out	*ln*	The line structure to be initialized.

4.3.2.4 void structInitBtn (btn ∗ *ui_button*)

Initilaises a button structure

For use when declaring a button to ensure that all elements have defined values and predictable behavior.

Parameters

out	*ui_button*	The button structure to be initialized.

4.3.2.5 void structInitTurtleState (turtle_state ∗ *turtle*)

Initilaises a turtle_state structure

For use when declaring a turtle_state structure to ensure that all elements have defined values and predictable behavior.

Parameters

`out`	*turtle*	The turtle structure to be initialized.

4.4 src/turtle.c File Reference

A source file for functions used to convert lsystem strings into arrays of coordinate pairs that make up the lines to be drawn.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "structs.h"
#include "turtle.h"
#include "lsys.h"
```

Functions

- int stringToTurtle (lsystem ∗lsys)
- void savePos (turtle_state ∗∗root, turtle_state current_turtle)
- void popPos (turtle_state ∗∗root, turtle_state ∗current_turtle)
- int penDownLine (turtle_state ∗current_turtle, int length, int line_list_pos, lsystem ∗lsys)
- void penUpLine (turtle_state ∗current_turtle, int length)
- int countMoves (char ∗string)

4.4.1 Detailed Description

A source file for functions used to convert lsystem strings into arrays of coordinate pairs that make up the lines to be drawn.

The sting is counted for characters which will produce a visible line and allocate memory for an array of lines of the required length, and then the program will run through the string again, updating the turtle as it goes and and add a pair of coordinates to the line list every time a line drawing character is encountered.

4.4.2 Function Documentation

4.4.2.1 int stringToTurtle (lsystem ∗ *lsys*)

The main function for converting the string produced by an lsystem into a list of coordinate pairs for lines to be drawn between.

This is done by firstly runnung through the string that defines the pattern to be drawn and counting the number of lines that will need to be drawn. This length is then used to allocate Memory for an array of lines. A turtle is then introduced and the string is gone through again. The turtle is kept updated and everytime it draws a line, one is added to the line list and if the position needs to be stored or retrieved from the turtle stack then it extracts or adds to the top element of the linked list.

Parameters

| out | *lsys* | pointer to the lsystem (needed for angle incriment and line length). |

Returns

the number of coordinate pairs that are held in the line_list.

4.4.2.2 void savePos (turtle_state ∗∗ *root,* turtle_state *current_turtle*)

Saves the current position of the turtle to the top of the linked list.

This is done by initialising a new turtle state and filling it with the current_turtle data. This is then set to point to the top of the turtle stack and the previous pointer to the top is then set to point to the new top.

Parameters

| out | *root* | ponter pointer to the root of the chain of positions. |
| in | *current_turtle* | the currentky active turtle. |

4.4.2.3 void popPos (turtle_state ∗∗ *root,* turtle_state ∗ *current_turtle*)

pops off the top of the chain of saved turtle states, changing where the root points and freeing the removed data.

This is done by copying the data from the top of the turtle stack to the current_turtle, then setting a temporary pointet to point to the second element in the stack. The top of the stack is then freed and the pointer that wa pointing to the top is now set to the temporary pointer.

Parameters

| out | *root* | pointer pointer to the top of the turtle stack. |
| out | *current_turtle* | pointer to the currently active turtle. |

4.4.2.4 int penDownLine (turtle_state ∗ *current_turtle,* int *length,* int *line_list_pos,* lsystem ∗ *lsys*)

Adds a line to the line list and updates the turtles position.

This function uses some simple trigonometry to calculate the final position of the turtle after drawing a line of a given length, from a given starting point at a given heading. The starting position and end position are then added to the line list as a pair of coordinates.

Parameters

out	*current_turtle*	pointer to the currently active turtle.
out	*line_list*	pointer pointer to the array of coordinate pairs for drawing lines.
in	*length*	length of each line to be drawn
in	*linelist_pos*	an integer that holds the current position of the next empty element in the line list.

Returns

the current position in the line list.

4.4.2.5 void penUpLine (turtle_state ∗ *current_turtle,* int *length*)

Moves the turtle by the line length using the current heading, but without adding that line to the line list.

This is done using simple trigonometry to calculate the change in x and y using the given length anf heading. These values are then added to the starting position and then the turtle is updated to this new position.

Parameters

out	*current_turtle*	pointer to the current active turtle.
in	*length*	length of the distance to be moved.

4.4.2.6   int countMoves (char ∗ *string*)

Runs through the string and counts the number of line drawing characters.

Parameters

in	*string*	string that is going to be run through and counted.

4.5   src/ui.c File Reference

File containing source code for all UI functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "structs.h"
#include "lsys.h"
#include "turtle.h"
#include "ui.h"
```

Functions

- void createHomeScreenButtons (btn ∗screen_buttons, TTF_Font ∗font)
- void createOptionsScreenButtons (btn ∗screen_buttons, TTF_Font ∗font)
- void createDrawScreenButtons (btn ∗screen_buttons, TTF_Font ∗title_font, TTF_Font ∗body_font)
- void drawHomeScreen (SDL_Renderer ∗renderer, btn ∗screen_buttons, TTF_Font ∗title_font, TTF_Font ∗body_font)
- void drawOptionsScreen (SDL_Renderer ∗renderer, btn ∗screen_buttons, TTF_Font ∗title_font, TTF_Font ∗body_font, lsystem ∗lsys)
- void drawDrawingScreen (SDL_Renderer ∗renderer, btn ∗screen_buttons, TTF_Font ∗title_font, TTF_Font ∗body_font, lsystem ∗lsys)
- int homeScreenClick (SDL_Renderer ∗renderer, SDL_Event event, btn ∗button_list, int win_flag)
- int optionsScreenClick (SDL_Renderer ∗renderer, SDL_Event event, btn ∗button_list, int win_flag, lsystem ∗lsys)
- int drawScreenClick (SDL_Renderer ∗renderer, SDL_Event event, btn ∗button_list, int win_flag, lsystem ∗lsys, TTF_Font ∗title_font, TTF_Font ∗body_font)
- void createHomeButton (btn ∗screen, TTF_Font ∗font)
- void createBackButton (btn ∗screen, TTF_Font ∗font)
- void addButton (btn ∗button_array, coordinate pos, int width, int height, SDL_Colour colour, TTF_Font ∗font, char ∗text)
- void drawTextToRenderer (SDL_Renderer ∗renderer, int x_pos, int y_pos, char ∗text, TTF_Font ∗font, int allignment)
- void drawButtonToRenderer (SDL_Renderer ∗renderer, btn ui_button)
- void drawAllButtonsToRenderer (SDL_Renderer ∗renderer, btn ∗screen, int len)
- void drawColourChartToRenderer (SDL_Renderer ∗renderer, int x_pos, int y_pos)

- void drawInputColourBox (SDL_Renderer ∗renderer, SDL_Colour colour, int x_pos, int y_pos)
- void drawInfoToRenderer (SDL_Renderer ∗renderer, int x_pos, int y_pos, lsystem lsys, TTF_Font ∗title_font, TTF_Font ∗body_font)
- void drawLine (SDL_Renderer ∗renderer, coordinate start, coordinate end, int x_max, int x_min, int y_max, int y_min)
- void drawFractal (SDL_Renderer ∗renderer, int length, line ∗line_list, SDL_Colour line_colour)
- void printRule (SDL_Renderer ∗renderer, int x_pos, int y_pos, char ∗original, char ∗replacement, TTF_Font ∗font)
- int clickInButton (SDL_Event event, btn screen_button)
- SDL_Colour getPixelColour (SDL_Renderer ∗renderer, int win_width, int win_height, int x_pos, int y_pos)
- void drawBG (SDL_Renderer ∗renderer)
- void imgSave (SDL_Renderer ∗renderer, lsystem ∗lsys)
- void sequenceSave (SDL_Renderer ∗renderer, lsystem ∗lsys, TTF_Font ∗title_font, TTF_Font ∗body_font)

4.5.1 Detailed Description

File containing source code for all UI functions.

The first half of the functions defined in the file are specific to the drawsystem program, containing instructions on creating, drawing and handling clicks on the various screens used by the program. The second half of the file contains genral UI functions used by the program specific functions.

4.5.2 Function Documentation

4.5.2.1 void createHomeScreenButtons (btn ∗ screen_buttons, TTF_Font ∗ font)

Defines all of the buttons that will appear on the home screen.

An array of buttons is created that can be passed to both the screen drawing functions and the click handeling functions so that they are both looking at the same array of buttons. This makes updating, changing and adding to the buttons that appear on each screen much easier.

Parameters

out	screen_buttons	a structure that will hold information on buttons to appear on the home screen.
in	font	font for the button lables.

4.5.2.2 void createOptionsScreenButtons (btn ∗ screen_buttons, TTF_Font ∗ font)

Defines all of the buttons that will appear on the options screen.

An array of buttons is created that can be passed to both the screen drawing functions and the click handeling functions so that they are both looking at the same array of buttons. This makes updating, changing and adding to the buttons that appear on each screen much easier.

Parameters

out	screen_buttons	a structure that will hold infromation on buttons to appear on the options screen.
in	font	font for the button lables.

4.5.2.3 void createDrawScreenButtons (btn ∗ screen_buttons, TTF_Font ∗ title_font, TTF_Font ∗ body_font)

Defines all of the buttons that will appear on the options screen.

An array of buttons is created that can be passed to both the screen drawing functions and the click handeling functions so that they are both looking at the same array of buttons. This makes updating, changing and adding to the buttons that appear on each screen much easier.

Parameters

screen_buttons	an array that will hold infromation on buttons to appear on the draw screen.
title_font	large font to be used on some buttons.
body_font	small font to be used on some buttons.

4.5.2.4 void drawHomeScreen (SDL_Renderer ∗ *renderer,* btn ∗ *screen_buttons,* TTF_Font ∗ *title_font,* TTF_Font ∗ *body_font*)

Draws the home sceen to the renderer.

draws the background colour to the renderer, then all the buttons defined in the btn array and then some title text.

Parameters

out	renderer	renderer for screen to be drawn to.
in	screen_buttons	buttons to be drawn to the screen.
in	title_font	font for the title text
in	body_font	font for the body text.

4.5.2.5 void drawOptionsScreen (SDL_Renderer ∗ *renderer,* btn ∗ *screen_buttons,* TTF_Font ∗ *title_font,* TTF_Font ∗ *body_font,* lsystem ∗ *lsys*)

Draws the options screen to the renderer.

Sraws the background, then some title text, then colour charts for picking drawing colours, then the buttons defined in the btn array and finally some text to indicate the current L-System loaded into the lsystem structre.

Parameters

out	renderer	renderer for screen to be drawn to.
in	screen_buttons	buttons to be drawn to the screen.
in	title_font	font to be used for the title text.
in	body_font	font to be used for the body text.
out	lsys	lsystem container that has drawing information to be changed byt options buttons.

4.5.2.6 void drawDrawingScreen (SDL_Renderer ∗ *renderer,* btn ∗ *screen_buttons,* TTF_Font ∗ *title_font,* TTF_Font ∗ *body_font,* lsystem ∗ *lsys*)

Sraws the drawing screen to the renderer.

Starts by drawing the background to the renderer, then buttons, instruction text and finally the lsystem drawign window. The drawing of the window runs the functions defined in lsys.c and turtle.c if the relevant flags are set to create the string, and then the line list that are required to draw the lsystem.

To allow the buttons to update the screen live the lsystem needs to be redrawn each frame. To keep the speed up and minimize the number of calultions being done by the computer flags are set to tell the program when the string and line list need to be recalulated and when it can just redraw the last ones that is used (which are stored in the lsystem stucture).

Parameters

out	renderer	renderer for the screen to be drawn to.
in	screen_buttons	buttons to be drawn to the screen.

in	*title_font*	font to be used for the title text.
in	*body_font*	font to be used for the body text.
out	*lsys*	lsystem that contains information to be drawn to the screen.

4.5.2.7   int homeScreenClick (SDL_Renderer ∗ *renderer,* SDL_Event *event,* btn ∗ *button_list,* int *win_flag*)

Handles what happens when a button is clicked by checking the position against buttons defined in the button list.

The position of the click is checked against each button in the array and if it is inside one of these then the relevant value is returned.

Parameters

in	*event*	a click event that contains infromation about which mouse button was pressed and where.
in	*win_flag*	a flag that indicates which window is currently being sent to the screen (main menu, creation options, display lsystem ect.).
in	*button*	an array of the buttons that are on the screen.
in	*win_flag*	the old window flag.

Returns

the new win_flag that will indicate what the new window should be (default is the same value that came in).

4.5.2.8   int optionsScreenClick (SDL_Renderer ∗ *renderer,* SDL_Event *event,* btn ∗ *button_list,* int *win_flag,* lsystem ∗ *lsys*)

Handles what happens when a button is clicked by checking the position against buttons on the current screen (which is indicated by the win_flag).

The position of the click is checked against each button in the array and if it is inside one of these then the relevant value is returned and changes made to the lsystem structure.

Parameters

in	*event*	a click event that contains infromation about which mouse button was pressed and where.
in	*win_flag*	a flag that indicates which window is currently being sent to the screen (main menu, creation options, display lsystem ect.).
in	*button_list*	an array of the buttons that are on the screen.
in	*win_flag*	the old window flag.
in	*lsys*	container for information about the lsystem, which will be edited depending on the options chosen.

Returns

the new win_flag that will indicate what the new window should be (default is the same value that came in).

4.5.2.9   int drawScreenClick (SDL_Renderer ∗ *renderer,* SDL_Event *event,* btn ∗ *button_list,* int *win_flag,* lsystem ∗ *lsys,* TTF_Font ∗ *title_font,* TTF_Font ∗ *body_font*)

Handles what happens when a button is clicked by checking the position against buttons on the current screen (which is indicated by the win_flag).

the position of the click is checked against each button in the array and if it is inside one of these then the relevant value is returned and flags set.

Parameters

in	*event*	a click event that contains infromation about which mouse button was pressed and where.
in	*win_flag*	a flag that indicates which window is currently being sent to the screen (main menu, creation options, display lsystem ect.).
in	*button*	an array of the buttons that are on the screen.
in	*win_flag*	the old window flag.
out	*lsys*	container for information about the lsystem, which will be edited depending on the options chosen.
in	*tile_font*	large font used in the save sequence button.
in	*body_font*	small font used in the save sequence button.

Returns

the new win_flag that will indicate what the new window should be (default is the same value that came in).

4.5.2.10 void createHomeButton (btn ∗ *screen,* TTF_Font ∗ *font*)

Adds a home button to the input btn array which makes up a screen.

Every screen will have a home button in the same place that looks the same to help make the program feel continuous.

Parameters

out	*screen*	the screen that the home button is to be added to.
in	*font*	font for the text on the home button.

4.5.2.11 void createBackButton (btn ∗ *screen,* TTF_Font ∗ *font*)

Adds a back button to the input screen.

Every screen will have a back button in the same place that looks the same to help make the program feel continuous.

Parameters

out	*screen*	the screen that the back button is to be added to.
in	*font*	font for the text on the home button.

4.5.2.12 void addButton (btn ∗ *button_array,* coordinate *pos,* int *width,* int *height,* SDL_Colour *colour,* TTF_Font ∗ *font,* char ∗ *text*)

A function that sets the variable for the button structure taken as input.

Parameters

out	*button*	button that is to hold the information.
in	*pos*	position of the top left corner of the button.
in	*width*	width of the button.
in	*height*	height of the button.
in	*colour*	colour of the button.

in	*font*	font for the button text.
in	*text*	string containing the lable for the button.

4.5.2.13   void drawTextToRenderer (SDL_Renderer * *renderer,* int *x_pos,* int *y_pos,* char * *text,* TTF_Font * *font,* int *allignment*)

Wraps up all of the SDL and SDL_ttf functions required for drawing to the renderer and brings them together into an easy to use function.

To draw text to the renderer, the text must be created as a surface, then copied to a texture which can then be copied to a renderer inside a rectangle that is the same size as the texture. This function wraps this proces up and also allows a flag to be set for text allignment to make positioning easier for the user.

Parameters

out	*renderer*	pointer to the renderer that will hold the output.
in	*x_pos*	x position for the text on the renderer.
in	*y_pos*	y position for the text on the renderer.
in	*text*	character array for the text to be written to the renderer.
in	*font*	the font for the text to be written.
in	*allignment*	an option for thext allignment, 0 is centered, 1 is left alligned and 2 is right alligned in relation to the input coorinates.

4.5.2.14   void drawButtonToRenderer (SDL_Renderer * *renderer,* btn *ui_button*)

draws a single button to the renderer

Parameters

out	*renderer*	The renderer that the button is to be drawn to.
in	*ui_button*	a structure that holds information on the button to be drawn.

4.5.2.15   void drawAllButtonsToRenderer (SDL_Renderer * *renderer,* btn * *screen,* int *len*)

Runs through the button list that mekes up a screen and prints each one to the renderer if it exists.

Parameters

out	*renderer*	renderer for buttons to be drawn to.
in	*screen*	array contianing buttons to be drawn to the renderer.
in	*len*	number of buttons in the btn array.

4.5.2.16   void drawColourChartToRenderer (SDL_Renderer * *renderer,* int *x_pos,* int *y_pos*)

Draws a colour selection chart to the rendenderer.

To crate a colour selection chart for the user to use, red, green and blue variables are initialised, and then cycled through in multiples of 51 in nested for loops to create a distribution of colours. This creates 6 different values for each colour chanel, resulting in 216 colours to chooe from. Each colour is then drawn in its own box on the renderer and after evey 6 blocks a new line is started.

Parameters

out	*renderer*	pointer to the renderer that is to be drawn to.
in	*x_pos*	x position of the top left-hand corner of the chart.
in	*y_pos*	y position of the top left_hand corner of the chart.

4.5.2.17   void drawInputColourBox (SDL_Renderer ∗ *renderer,* **SDL_Colour** *colour,* **int** *x_pos,* **int** *y_pos* **)**

A function for drawing a coloured box to the screen.

Used for displaying the current line and background colours before drawing the fractal to the renderer.

Parameters

out	*renderer*	renderer for box to be drawn to.
in	*colour*	colour that the box will be.
in	*x_pos*	x position for the top left corner of the box.
in	*y_pos*	y position for the top left corner of the box.

4.5.2.18   void drawInfoToRenderer (SDL_Renderer ∗ *renderer,* **int** *x_pos,* **int** *y_pos,* **lsystem** *lsys,* **TTF_Font** ∗ *title_font,*
TTF_Font ∗ *body_font* **)**

displays information about the lsystem being drawn to the screen.

If the display_info flag is true, then information about the lsystem will be drawn to the renderer as well as the fractal
its self. This info will also be present in the saved images if it is on the screen at the time of the save button being
pressed.

Parameters

out	*renderer*	renderer for information to be drawn to.
in	*x_pos*	x coordinate of top left corner of info box.
in	*y_pos*	y coordinate of top left corner of info box.
in	*lsys*	lsystem containing information to be printed.
in	*title_font*	large font.
in	*body_font*	small font.

4.5.2.19   void drawLine (SDL_Renderer ∗ *renderer,* **coordinate** *start,* **coordinate** *end,* **int** *x_max,* **int** *x_min,* **int** *y_max,* **int**
y_min **)**

A line drawing function that implements the bresenheim line drawing algorithm to draw a line between two points.

Based on source code from Eike Anderson, Computing for Graphics level c, lecture 19, 2015. Edited to work with
struct Coordinate as start and end points for line. I felt it was necessary to write my own (as opposed to using th
SDL_RederDrawLine() function) as I could therefor check the pixels being drawn to on a point by point basis so that
I could constrain the drawing to a box on the screen while not having to miss out whole lines.

Parameters

out	*renderer*	target to be rendered to.
in	*start*	start point for the line.
in	*end*	end point of the line.
in	*x_max*	maximum x coordinate to draw to.
in	*x_min*	minimum x coordinate to draw to.
in	*y_max*	maximum y coordinate to draw to.

in	*y_min*	minimum y coordinate to draw to.

4.5.2.20 void drawFractal (SDL_Renderer ∗ *renderer,* int *length,* **line ∗ *line_list,* SDL_Colour *line_colour*)**

Draws the fractal storred as int line_list which is of length 'length'

Parameters

out	*renderer*	renderer to be drawn to.
in	*length*	number of items in the line list.
in	*line_list*	a list of coordinate pairs whcih define the lines that need to be drawn.
in	*line_colour*	colour for the lines to be drawn.

4.5.2.21 void printRule (SDL_Renderer ∗ *renderer,* int *x_pos,* int *y_pos,* char ∗ *original,* char ∗ *replacement,* TTF_Font ∗ *font*)

Formats and writes out the rule for the input character.

For printing the rules to the drawing screen, the strings needed to be specifically formatted and this function wraps up that formatting.

Parameters

out	*renderer*	renderer for the rule to be written to.
in	*x_pos*	x position for the top left hand corner of the text.
in	*y_pos*	y position of the top left hand corner of the text.
in	*original*	the character that is to be replaced in a string format.
in	*replacement*	the replacement string for that character.
in	*body_font*	the font to be used.

4.5.2.22 int clickInButton (SDL_Event *event,* **btn *screen_button*)**

checks to see if a click happened inside a specific button.

Uses the button to calulate upper and lower boundaries for x and y positions that fall inside the button and then gets the click event data and compares the position to these limits. Includes a safety check to make sure that the event is a click event.

Parameters

in	*event*	the click event in question.
in	*screen_button*	the button that contains the information on the position of the window.

Returns

> true (1) if click is inside the button, fase (0) if not.

4.5.2.23 SDL_Colour getPixelColour (SDL_Renderer ∗ *renderer,* int *win_width,* int *win_height,* int *x_pos,* int *y_pos*)

Finds the pixel colour on the renderer of the point given by x_pos, y_pos.

Based on source code from Eike Anderson, Computing for Graphics level c,

1. Edited to work with struct SDL_Colour. It works by firstly ckecking to make sure that the click was inside the window, then, if it was, a helper surface is created and then reformatted to take the information retrieved from

the renderer using SDL_RenderReadPixels. The RGB values from this data is then copied into the p_colour variable that is then returned.

Parameters

in	*renderer*	renderer that holds the pixel information.
in	*win_width*	width of the window (used to make sure the pixel piosition exists).
in	*win_height*	height of the window (used to make sure the pixel position exists).
in	*x_pos*	x coordinate of the pixel being looked at.
in	*y_pos*	y coordinate of the pixel beign looked at.

Returns

the colour of the pixel in rgba format.

4.5.2.24 void drawBG (SDL_Renderer ∗ *renderer*)

Draws background to renderer.

clears the background to a single colour and then draws a menu bar in a different colour to the top of the renderer.

Parameters

out	*renderer*	renderer for the bakkground to be drawn to

4.5.2.25 void imgSave (SDL_Renderer ∗ *renderer,* lsystem ∗ *lsys*)

Saves the fractal image as a single bmp image.

Copies the drawing region of the renderer to a surface to be saved by the SDL_SaveBMP() function.

Parameters

in	*renderer*	renderer that contains onformation to be copied
in	*lsys*	the structure that contains the information for naming

4.5.2.26 void sequenceSave (SDL_Renderer ∗ *renderer,* lsystem ∗ *lsys,* TTF_Font ∗ *title_font,* TTF_Font ∗ *body_font*)

Saves the fractal as a sequence of bmp images that show the fractal being drawn line by line.

clears the renderer before iteratively drawing lines to it, then converting to surface and saving. The program splits up the lines being drawn so that there is a limit of 100 frames that can be saved. this keeps the program running quickly but limits what the user can do with the output. I decided on doing it this way because the save sequence is error porne and slow (moving the window while saving off a sequence causes artifacts).

Parameters

in	*lsys*	structure that contians information for drawing img sequence.

Index