Gecko Battleships

version 1

Generated by Doxygen 1.9.7

1 Gecko Battleships Linux control program documentation

1.1 Specification

This program communicates with a Silicon Labs EFM32 STK3700 Giant Gecko developer board via a serial port. The board runs a Battleships game, where the player can shoot at the enemy ships by pressing the keys on the keyboard. The game is displayed on the Gecko board's LCD screen. The program reads the key presses from the standard input and sends them to the board via the serial port. The program also reads the data sent by the board via the serial port and prints the game state to a csv file. At the end of the game the program plots the score against the number of steps taken using the gnuplot program.

1.2 Usage

The program can be run from the command line with the following options:

- -h: Print help message to the standard output containing the command line options and the game controls.
- -s d=device file,s=baud rate: Set the device file and baud rate for the serial port. The device file must be a valid path to a serial port device file, e.g. /dev/ttyASM0. The baud rate must be a valid baud rate for the serial port, e.g. 9600.
- -o file: Set the game data output file name. The file name must be a valid path to a file, e.g. game.log. The file will be overwritten if it already exists. If no options are given, the program will use the default values: device file: /dev/ttyASM0, baud rate: 9600, output file: game.log.

1.3 Controls

The game controls are the following:

- WASD keys: Move the cursor on the LCD screen.
- E key: Shoot at the current cursor position.
- R key: Reset the game and the control program.
- Q key: Quit control program. This does not reset the game.
- . (dot) key: Win game instantly. This does not reveal the enemy ships, only triggers the win condition on the Gecko board.

1.4 Compilation

The program can be compiled with GCC using the following command: gcc -i battleships.c -o battleships

2 File Documentation

2.1 battleships.c File Reference

Function implementations for the Gecko Battleships program.

Macros

- #define CMDLINE_DBG true

  Print debug messages to the terminal.

Functions

- void exit_function (void)

  Exit function set by atexit() to reset the terminal to the original settings. Called when the program exits normally, at errors, or by a signal.
- bool set_g_speed (int speed)

  Set the speed of the serial port for communicating with the Gecko board.
- void print_messages (int message_type)

  Print colorful info, error, warning, debug and success messages in brackets to the standard output.
- void print_help (void)

  Print help message to the standard output containing the command line options and the game controls.
- int terminal_setup (bool reset)

  Set up the terminal for non-canonical input and output and disable echoing of input characters.
- void plot_results (void)

  Print a plot showing the score against the steps taken. Uses the gnuplot program.
- int main (int argc, char ∗argv[ ])

  Main function of the program.

Variables

- uint32_t g_serial_speed = 9600
  Default serial port baud rate.
- char g_serial_device [CFGSTR_SIZE] = ¨/dev/ttyACM0¨
  Default serial port device path.
- char g_outfile [FILENAME_SIZE] = ¨game.log¨
  Default output file name.
- struct BNUM_speed g_speed [ ]
  Constants from /usr/include/asm-generic/termbits.h.

2.1.1 Detailed Description

Function implementations for the Gecko Battleships program.

Author

Botond Piller (`p.botondakos@edu.bme.hu`)

Version

1

Date

2023-06-08

Copyright

Copyright (c) 2023

2.1.2 Macro Definition Documentation

CMDLINE_DBG

```
#define CMDLINE_DBG true
```

Print debug messages to the terminal.

Debug messages include information about the program's state, such as the successful completion of a function, variable values, serial communication status, etc. Set to false to get cleaner output.

2.1.3 Function Documentation

exit_function()

```
void exit_function (
            void )
```

Exit function set by atexit() to reset the terminal to the original settings. Called when the program exits normally, at errors, or by a signal.

This function is called by atexit() when the program exits normally, at errors, or by a signal. It resets the terminal to the original settings and flushes the output file stream.

main()

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

Main function of the program.

This function implements most of the program's functionality, as follows:

- It parses and interprets command line arguments (-h, -s d=device file,s=baud rate, -o file) using the getopt() function in a while loop.
- It calls the terminal_setup() function to set up the terminal for non-canonical input and output and to disable echoing of input characters.
- The step and score integer variables get initialized to a value of 0, these will store the number of steps taken and the score, respectively.
- It creates a termios structure for the serial port communication with the Gecko board and fills the structure with the desired settings (8 bits, non-canonical read, 1/2 second timeout). The speed gets set by the set_g_speed() function. The open() function is called to open the device file for reading and writing.
- If the serial communiation is set up successfully, the character 'r' gets sent to reset the game.
- It opens the output file for writing: if the file already exists, it gets overwritten.
- The time of the start of the game gets stored in the start_time variable.
- The linebuf_in and linebuf_out character arrays get initialized to store data read from the standard input and the serial port, respectively.
- The pfds pollfd structure gets initialized to monitor the standard input and the serial port for data to read. The program enters a while() loop and polls the file descriptors as long as the game doesn't end. If the standard input is ready to read, the program reads the data into the linebuf_in array, then writes it to the serial port. If the character read is 'q', the program terminates. If the character read is 'r', the program resets the game by sending the character 'r' to the serial port, triggering software reset on the Gecko board and resetting the score, steps taken and start time.
- If the serial port is ready to read, the program reads the data into the linebuf_out array. If the character read is '1', there was a hit in the game, so the score gets incremented. If the character read is '0', the last shot missed, only the step counter gets incremented. In both cases a line gets written to the output file containing the time, the steps taken and the score.
- If the character received is 'x', the game ended, the program prints the steps taken and the time elapsed to the standard output. It then calls the plot_results() function to plot the score against the steps taken and exits the program.

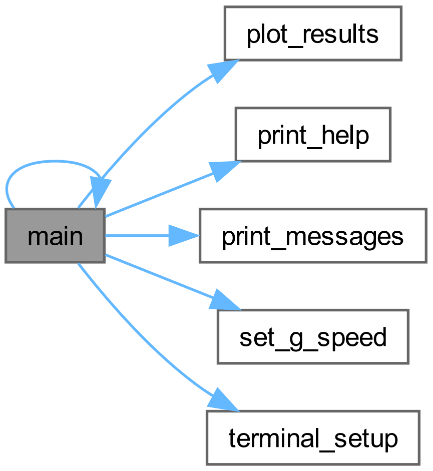
Parameters

argc	Argument count
argv	Argument vector

Returns

int 0 after successful execution.

Here is the call graph for this function:



plot_results()

```
void plot_results (
             void  )
```

Print a plot showing the score against the steps taken. Uses the gnuplot program.

The function sets the gnuplotPipe file pointer to a pipe stream connected to the gnuplot program using the popen() function. It then writes the gnuplot commands to the pipe stream to plot the score against the steps taken.

print_help()

```
void print_help (
             void  )
```

Print help message to the standard output containing the command line options and the game controls.

print_messages()

```
void print_messages (
             int message_type )
```

Print colorful info, error, warning, debug and success messages in brackets to the standard output.

Supported message types: MESSAGE_INFO, MESSAGE_WARNING, MESSAGE_ERROR, MESSAGE_DEBUG, MESSAGE_SUCCESS. If the message type is not supported, it returns without printing anything.

Parameters

message_type	Type of message to print.

set_g_speed()

```
bool set_g_speed (
             int speed )
```

Set the speed of the serial port for communicating with the Gecko board.

Iterates through the g_speed BNUM_speed array until it finds the desired speed parameter. Writes the speed in the g_termio_speed global variable. If the desired speed is not found, it returns false.

Parameters

speed	Baud rate to set

Returns

true Speed set successfully.

false Invalid speed, could not set.

terminal_setup()

```
int terminal_setup (
             bool reset )
```

Set up the terminal for non-canonical input and output and disable echoing of input characters.

Creates two static termios structures: one for the original terminal settings and one for the new settings. On the first run, it saves the original terminal settings to the oldt structure and copies it to the newt structure, setting the ICANON and ECHO flags to 0. ICANON normally takes care that one line at a time will be processed that means it will return if it sees a ¨\n¨ or an EOF or an EOL, by disabling ICANON the program will read in everything that is being input, regardless of newlines or end-of-input markers.ECHO causes each key typed to be printed to the terminal. By disabling this, the program will not print the characters you type to the screen. On subsequent runs checks the value of the reset parameter. If reset is true, it restores the original terminal settings from the oldt structure. If reset is false, it sets the new terminal settings from the newt structure.

Parameters

reset	If true, reset the terminal to the original settings.

Returns

int 0 if successful.

2.1.4 Variable Documentation

g_outfile

```
char g_outfile[FILENAME_SIZE] = ¨game.log¨
```

Default output file name.

The game data is written to this file in the following csv format: ¨timestamp, score, steps\n¨. The file is created in the current working directory with the name specified here (¨./game.log¨).

g_serial_device

```
char g_serial_device[CFGSTR_SIZE] = ¨/dev/ttyACM0¨
```

Default serial port device path.

By default, the program attempts to open the serial port at ¨/dev/ttyACM0¨ for communication with the Gecko board. This can be changed with the ¨-s d=...¨ command line option.

g_serial_speed

```
uint32_t g_serial_speed = 9600
```

Default serial port baud rate.

By default the program attempts to open the serial port at 9600 baud. This can be changed with the ¨-s s=...¨ command line option.

g_speed

```
struct BNUM_speed g_speed[]
```

Constants from /usr/include/asm-generic/termbits.h.

Constants are octal in termbits.h, they are converted to decimal here. Contains all baud rates supported by the Linux kernel.

2.2 battleships.h File Reference

Header file for the Gecko Battleships program.

Macros

- #define CFGSTR_SIZE 64

  Size of device path string.
- #define FILENAME_SIZE 1024

  Size of output file name string.
- #define BUFLEN (32)

  Length of line buffer for serial communication.
- #define PFDSLEN (2)

  Number of file descriptors to monitor.

Functions

- bool set_g_speed (int speed)

  Set the speed of the serial port for communicating with the Gecko board.
- void print_messages (int message_type)

  Print colorful info, error, warning, debug and success messages in brackets to the standard output.
- void print_help (void)

  Print help message to the standard output containing the command line options and the game controls.
- int terminal_setup (bool reset)

  Set up the terminal for non-canonical input and output and disable echoing of input characters.
- void exit_function (void)

  Exit function set by atexit() to reset the terminal to the original settings. Called when the program exits normally, at errors, or by a signal.
- void plot_results (void)

  Print a plot showing the score against the steps taken. Uses the gnuplot program.
- int main (int argc, char ∗argv[ ])

  Main function of the program.

2.2.1 Detailed Description

Header file for the Gecko Battleships program.

Author

Botond Piller (`piller.botond.akos@edu.bme.hu`)

Version

1

Date

2023-06-08

Copyright

Copyright (c) 2023

2.2.2 Macro Definition Documentation

BUFLEN

```
#define BUFLEN (32)
```

Length of line buffer for serial communication.

CFGSTR_SIZE

```
#define CFGSTR_SIZE 64
```

Size of device path string.

FILENAME_SIZE

```
#define FILENAME_SIZE 1024
```

Size of output file name string.

PFDSLEN

```
#define PFDSLEN (2)
```

Number of file descriptors to monitor.

2.2.3 Function Documentation

exit_function()

```
void exit_function (
            void  )
```

Exit function set by atexit() to reset the terminal to the original settings. Called when the program exits normally, at errors, or by a signal.

This function is called by atexit() when the program exits normally, at errors, or by a signal. It resets the terminal to the original settings and flushes the output file stream.

main()

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

Main function of the program.

This function implements most of the program's functionality, as follows:

- It parses and interprets command line arguments (-h, -s d=device file,s=baud rate, -o file) using the getopt() function in a while loop.
- It calls the terminal_setup() function to set up the terminal for non-canonical input and output and to disable echoing of input characters.
- The step and score integer variables get initialized to a value of 0, these will store the number of steps taken and the score, respectively.
- It creates a termios structure for the serial port communication with the Gecko board and fills the structure with the desired settings (8 bits, non-canonical read, 1/2 second timeout). The speed gets set by the set_g_speed() function. The open() function is called to open the device file for reading and writing.
- If the serial communiation is set up successfully, the character 'r' gets sent to reset the game.
- It opens the output file for writing: if the file already exists, it gets overwritten.
- The time of the start of the game gets stored in the start_time variable.
- The linebuf_in and linebuf_out character arrays get initialized to store data read from the standard input and the serial port, respectively.
- The pfds pollfd structure gets initialized to monitor the standard input and the serial port for data to read. The program enters a while() loop and polls the file descriptors as long as the game doesn't end. If the standard input is ready to read, the program reads the data into the linebuf_in array, then writes it to the serial port. If the character read is 'q', the program terminates. If the character read is 'r', the program resets the game by sending the character 'r' to the serial port, triggering software reset on the Gecko board and resetting the score, steps taken and start time.
- If the serial port is ready to read, the program reads the data into the linebuf_out array. If the character read is '1', there was a hit in the game, so the score gets incremented. If the character read is '0', the last shot missed, only the step counter gets incremented. In both cases a line gets written to the output file containing the time, the steps taken and the score.
- If the character received is 'x', the game ended, the program prints the steps taken and the time elapsed to the standard output. It then calls the plot_results() function to plot the score against the steps taken and exits the program.

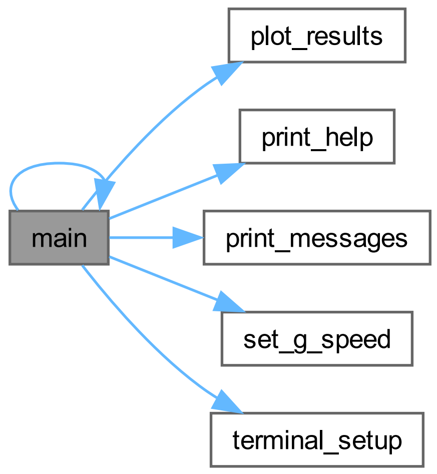
Parameters

argc	Argument count
argv	Argument vector

Returns

int 0 after successful execution.

Here is the call graph for this function:



plot_results()

```
void plot_results (
             void  )
```

Print a plot showing the score against the steps taken. Uses the gnuplot program.

The function sets the gnuplotPipe file pointer to a pipe stream connected to the gnuplot program using the popen() function. It then writes the gnuplot commands to the pipe stream to plot the score against the steps taken.

print_help()

```
void print_help (
             void  )
```

Print help message to the standard output containing the command line options and the game controls.

print_messages()

```
void print_messages (
             int message_type )
```

Print colorful info, error, warning, debug and success messages in brackets to the standard output.

Supported message types: MESSAGE_INFO, MESSAGE_WARNING, MESSAGE_ERROR, MESSAGE_DEBUG, MESSAGE_SUCCESS. If the message type is not supported, it returns without printing anything.

Parameters

message_type	Type of message to print.

set_g_speed()

```
bool set_g_speed (
             int speed )
```

Set the speed of the serial port for communicating with the Gecko board.

Iterates through the g_speed BNUM_speed array until it finds the desired speed parameter. Writes the speed in the g_termio_speed global variable. If the desired speed is not found, it returns false.

Parameters

speed	Baud rate to set

Returns

true Speed set successfully.

false Invalid speed, could not set.

terminal_setup()

```
int terminal_setup (
             bool reset )
```

Set up the terminal for non-canonical input and output and disable echoing of input characters.

Creates two static termios structures: one for the original terminal settings and one for the new settings. On the first run, it saves the original terminal settings to the oldt structure and copies it to the newt structure, setting the ICANON and ECHO flags to 0. ICANON normally takes care that one line at a time will be processed that means it will return if it sees a ¨\n¨ or an EOF or an EOL, by disabling ICANON the program will read in everything that is being input, regardless of newlines or end-of-input markers.ECHO causes each key typed to be printed to the terminal. By disabling this, the program will not print the characters you type to the screen. On subsequent runs checks the value of the reset parameter. If reset is true, it restores the original terminal settings from the oldt structure. If reset is false, it sets the new terminal settings from the newt structure.

Parameters

reset	If true, reset the terminal to the original settings.

Returns

int 0 if successful.

2.3 battleships.h

Go to the documentation of this file.

```c

// header guard
#ifndef SERIAL_H
#define SERIAL_H

// Standard C libraries
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <termios.h>
#include <sys/select.h>
#include <poll.h>

// Global variables to store configuration parameters
#define CFGSTR_SIZE 64
#define FILENAME_SIZE 1024

// ANSI escape codes for terminal colors
#define TERM_COLOR_RED "\033[0;31m"
#define TERM_COLOR_GREEN "\033[0;32m"
#define TERM_COLOR_YELLOW "\033[0;33m"
#define TERM_COLOR_BLUE "\033[0;34m"
#define TERM_COLOR_MAGENTA "\033[0;35m"
#define TERM_COLOR_DEFAULT "\033[0m"

// Message types for print_message()
#define MESSAGE_INFO 0
#define MESSAGE_WARNING 1
#define MESSAGE_ERROR 2
#define MESSAGE_DEBUG 3
#define MESSAGE_SUCCESS 4

#define BUFLEN (32)
#define PFDSLEN (2)

// Accessing termios.h Bnum type speed definition from the command line
struct BNUM_speed
{
    uint32_t speed;
    uint32_t bnum;
};

char *optarg; // To avoid VSCODE nagging about optarg not being declared.

bool set_g_speed(int speed);

void print_messages(int message_type);

void print_help(void);

int terminal_setup(bool reset);

void exit_function(void);

void plot_results(void);

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

#endif // SERIAL_H
```