Ingenieursproject II: Accelerometer

Jack The Lumberjack

This document contains the code documentation of the "Jack the Lumberjack" game. The code is meant to run on a Dwenguino with an accelerometer attached to it. The hardware configuration, together with the description of the game are documented in the "Verslag_Ingenieursproject_Antoine_Sebastiaan" file accompanying this document.

Authors

- Antoine De Schrijver
- Sebastiaan Verplancke

Disclaimer

This document was created using Doxygen 1.8.14

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 Game Struct Reference

Struct containing all game variables.

```
#include <dwenguinoLumberjack.h>
```

Data Fields

- int ** tree

    Array containing the approximate positions of the branches in the tree.
- int side_jack

    Keeps track of the side of the player (left or right).
- int score

    Keeps track of the score.
- int counter

    Gives the exact position of the branch (1 to 5).
- int difficulty

    Defines the rate at which an LED goes off.
- int state

    Gives the game state: ALIVE, DEAD or MENU.
- int tilt

    Determines the speed at which the branches fall down.
- int speed

    Rolling counter of the game. Caps at 29999, after that it goes back to 0 to prevent overflow.

3.1.1 Detailed Description

Struct containing all game variables.

3.1.2 Field Documentation

3.1.2.1 tree

```
int** tree
```

Array containing the approximate positions of the branches in the tree.

This variable is a 2D array consisting of two 1D arrays of length 8: they represent the left side and the right side of the tree.
These arrays are used to store the approximate positions of the branches in the tree. One element in the array contains 5 positions in the tree.
There can only be one branch in the total of the 5 positions. The exact position is stored in *counter* .

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

- dwenguinoLumberjack.h

3.2 lcd_info_type Struct Reference

Data Fields

- unsigned char **line**
- unsigned char **pos**

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

- dwenguinoLCD.h

Chapter 4

File Documentation

4.1 dwenguinoAccelerometer.c File Reference

```
#include "../HeaderFiles/dwenguinoAccelerometer.h"
#include <stdlib.h>
```

Functions

- void error ()

  Error function that is called on when initialization of the accelerometer fails. Prints an error to the LCD screen.
- void initAccelerometer ()

  Initializes the data-transfer between Dwenguino and accelerometer following the Master/Slave protocol.
- int read (int reg)

  Reads data from a specified register of the accelerometer following the Master/Slave protocol.
- int readX ()

  Reads the two registers containing data about the x-axis values and combines them.
- int readY ()

  Reads the two registers containing data about the y-axis values and combines them.
- int readZ ()

  Reads the two registers containing data about the z-axis values and combines them.
- int readPitch ()

  Uses the other read-functions to calculate the pitch angle at which the accelerometer is turned.
- int readRoll ()

  Uses the other read-functions to calculate the roll angle at which the accelerometer is turned.

4.1.1 Detailed Description

Author

Antoine, Sebastiaan

4.1.2 Function Documentation

4.1.2.1 error()

```
void error ( )
```

Error function that is called on when initialization of the accelerometer fails. Prints an error to the LCD screen.

This error serves only as a warning and doesn't exit the program. The player has to manually press the reset button.

Returns

Void

4.1.2.2 initAccelerometer()

```
void initAccelerometer ( )
```

Initializes the data-transfer between Dwenguino and accelerometer following the Master/Slave protocol.

If the initialization fails (probably due to faulty hardware), the *error* function is called.

Returns

Void

4.1.2.3 read()

```
int read (
             int reg )
```

Reads data from a specified register of the accelerometer following the Master/Slave protocol.

Parameters

reg	Register to read.

Returns

Register data.

4.1.2.4 readPitch()

```
int readPitch ( )
```

Uses the other read-functions to calculate the pitch angle at which the accelerometer is turned.

The formula for the pitch (in radians) is: $Pitch = atan\left(\frac{-X}{Z}\right)$

X is the value returned in *readX()*, *Z* is the value returned in *readZ()*

Returns

Pitch angle in degrees.

4.1.2.5 readRoll()

```
int readRoll ( )
```

Uses the other read-functions to calculate the roll angle at which the accelerometer is turned.

The formula for the roll (in radians) is: $Roll = atan\left(\frac{Y}{Z}\right)$

Y is the value returned in *readY()*, *Z* is the value returned in *readZ()*

Returns

Roll angle in degrees.

4.1.2.6 readX()

```
int readX ( )
```

Reads the two registers containing data about the x-axis values and combines them.

The register containing the most 8 significant bits of x is 0x3B, the one containing the 8 less significant bits is 0x3C.

Returns

X-axis accelerometer data.

4.1.2.7 readY()

```
int readY ( )
```

Reads the two registers containing data about the y-axis values and combines them.

The register containing the most 8 significant bits of y is 0x3D, the one containing the 8 less significant bits is 0x3E.

Returns

Y-axis accelerometer data.

4.1.2.8 readZ()

```
int readZ ( )
```

Reads the two registers containing data about the z-axis values and combines them.

The register containing the most 8 significant bits of z is 0x3F, the one containing the 8 less significant bits is 0x40.

Returns

Z-axis accelerometer data.

4.2 dwenguinoAccelerometer.h File Reference

This file contains all the functions that interact with the accelerometer.

```
#include "dwenguinoLCD.h"
#include "dwenguinoLumberjack.h"
```

Macros

- #define **LEFT** 1
- #define **RIGHT** 0

Functions

- void error ()
  Error function that is called on when initialization of the accelerometer fails. Prints an error to the LCD screen.
- void initAccelerometer ()
  Initializes the data-transfer between Dwenguino and accelerometer following the Master/Slave protocol.
- int read (int reg)
  Reads data from a specified register of the accelerometer following the Master/Slave protocol.
- int readX ()
  Reads the two registers containing data about the x-axis values and combines them.
- int readY ()
  Reads the two registers containing data about the y-axis values and combines them.
- int readZ ()
  Reads the two registers containing data about the z-axis values and combines them.
- int readPitch ()
  Uses the other read-functions to calculate the pitch angle at which the accelerometer is turned.
- int readRoll ()
  Uses the other read-functions to calculate the roll angle at which the accelerometer is turned.

4.2.1 Detailed Description

This file contains all the functions that interact with the accelerometer.

Author

Antoine, Sebastiaan

4.2.2 Function Documentation

4.2.2.1 error()

```
void error ( )
```

Error function that is called on when initialization of the accelerometer fails. Prints an error to the LCD screen.

This error serves only as a warning and doesn't exit the program. The player has to manually press the reset button.

Returns

Void

4.2.2.2 initAccelerometer()

```
void initAccelerometer ( )
```

Initializes the data-transfer between Dwenguino and accelerometer following the Master/Slave protocol.

If the initialization fails (probably due to faulty hardware), the *error* function is called.

Returns

Void

4.2.2.3 read()

```
int read (
            int reg )
```

Reads data from a specified register of the accelerometer following the Master/Slave protocol.

Parameters

reg	Register to read.

Returns

Register data.

4.2.2.4 readPitch()

```
int readPitch ( )
```

Uses the other read-functions to calculate the pitch angle at which the accelerometer is turned.

The formula for the pitch (in radians) is: $Pitch = atan\left(\frac{-X}{Z}\right)$

X is the value returned in *readX()*, *Z* is the value returned in *readZ()*

Returns

Pitch angle in degrees.

4.2.2.5 readRoll()

```
int readRoll ( )
```

Uses the other read-functions to calculate the roll angle at which the accelerometer is turned.

The formula for the roll (in radians) is: $Roll = atan\left(\frac{Y}{Z}\right)$

Y is the value returned in *readY()*, *Z* is the value returned in *readZ()*

Returns

Roll angle in degrees.

4.2.2.6 readX()

```
int readX ( )
```

Reads the two registers containing data about the x-axis values and combines them.

The register containing the most 8 significant bits of x is 0x3B, the one containing the 8 less significant bits is 0x3C.

Returns

X-axis accelerometer data.

4.2.2.7 readY()

```
int readY ( )
```

Reads the two registers containing data about the y-axis values and combines them.

The register containing the most 8 significant bits of y is 0x3D, the one containing the 8 less significant bits is 0x3E.

Returns

Y-axis accelerometer data.

4.2.2.8 readZ()

```
int readZ ( )
```

Reads the two registers containing data about the z-axis values and combines them.

The register containing the most 8 significant bits of z is 0x3F, the one containing the 8 less significant bits is 0x40.

Returns

Z-axis accelerometer data.

4.3 dwenguinoBoard.h File Reference

Initializes board.

```
#include <avr/io.h>
#include <avr/delay.h>
#include "dwenguinoLCD.h"
```

Macros

- #define **TRUE** 1
- #define **FALSE** 0
- #define **HIGH** 1
- #define **LOW** 0
- #define **PORT_HIGH** 0xFF
- #define **PORT_LOW** 0x00
- #define **INPUT** 0
- #define **OUTPUT** 1
- #define **SET_PIN_HIGH**(PORT, PIN) PORT |= (1 << PIN)
- #define **SET_PIN_LOW**(PORT, PIN) PORT &= ~(1 << PIN)
- #define **SET_BIT_HIGH**(REG, BIT) REG |= (1 << BIT)
- #define **SET_BIT_LOW**(REG, BIT) REG &= ~(1 << BIT)
- #define **BYTE** unsigned char
- #define **LEDS_DIR** DDRA

- #define **LEDS** PORTA
- #define **LED_ON**(LED) SET_PIN_HIGH(PORTA, LED)
- #define **LED_OFF**(LED) SET_PIN_LOW(PORTA, LED)
- #define **SW_C_HIGH** SET_PIN_HIGH(PORTC, 6)
- #define **SW_C_LOW** SET_PIN_LOW(PORTC, 6)
- #define **SW_C_IN** SET_PIN_LOW(DDRC, 6)
- #define **SW_C_OUT** SET_PIN_HIGH(DDRC, 6)
- #define **SW_W_HIGH** SET_PIN_HIGH(PORTE, 4)
- #define **SW_W_LOW** SET_PIN_LOW(PORTE, 4)
- #define **SW_W_IN** SET_PIN_LOW(DDRE, 4)
- #define **SW_W_OUT** SET_PIN_HIGH(DDRE, 4)
- #define **SW_S_HIGH** SET_PIN_HIGH(PORTE, 5)
- #define **SW_S_LOW** SET_PIN_LOW(PORTE, 5)
- #define **SW_S_IN** SET_PIN_LOW(DDRE, 5)
- #define **SW_S_OUT** SET_PIN_HIGH(DDRE, 5)
- #define **SW_E_HIGH** SET_PIN_HIGH(PORTE, 6)
- #define **SW_E_LOW** SET_PIN_LOW(PORTE, 6)
- #define **SW_E_IN** SET_PIN_LOW(DDRE, 6)
- #define **SW_E_OUT** SET_PIN_HIGH(DDRE, 6)
- #define **SW_N_HIGH** SET_PIN_HIGH(PORTE, 7)
- #define **SW_N_LOW** SET_PIN_LOW(PORTE, 7)
- #define **SW_N_IN** SET_PIN_LOW(DDRE, 6)
- #define **SW_N_OUT** SET_PIN_HIGH(DDRE, 6)
- #define **LCD_DATA** PORTA
- #define **LCD_DATA_DIR** DDRA
- #define **LCD_BACKLIGHT_ON** SET_PIN_HIGH(PORTE, 3)
- #define **LCD_BACKLIGHT_OFF** SET_PIN_LOW(PORTE, 3)
- #define **LCD_BACKLIGHT_OUT** SET_PIN_HIGH(DDRE, 3)
- #define **LCD_BACKLIGHT_IN** SET_PIN_LOW(DDRE, 3)
- #define **LCD_RW_HIGH** SET_PIN_HIGH(PORTE, 1)
- #define **LCD_RW_LOW** SET_PIN_LOW(PORTE, 1)
- #define **LCD_RW_OUT** SET_PIN_HIGH(DDRE, 1)
- #define **LCD_RS_HIGH** SET_PIN_HIGH(PORTE, 0)
- #define **LCD_RS_LOW** SET_PIN_LOW(PORTE, 0)
- #define **LCD_RS_OUT** SET_PIN_HIGH(DDRE, 0)
- #define **LCD_EN_HIGH** SET_PIN_HIGH(PORTE, 2)
- #define **LCD_EN_LOW** SET_PIN_LOW(PORTE, 2)
- #define **LCD_EN_OUT** SET_PIN_HIGH(DDRE, 2)
- #define **SERVO1** PORTC0
- #define **SERVO2** PORTC1
- #define **MOTOR1_0_HIGH** SET_PIN_HIGH(PORTC, 3)
- #define **MOTOR1_0_LOW** SET_PIN_LOW(PORTC, 3)
- #define **MOTOR1_1_HIGH** SET_PIN_HIGH(PORTC, 4)
- #define **MOTOR1_1_LOW** SET_PIN_LOW(PORTC, 4)
- #define **MOTOR2_0_HIGH** SET_PIN_HIGH(PORTC, 2)
- #define **MOTOR2_0_LOW** SET_PIN_LOW(PORTC, 2)
- #define **MOTOR2_1_HIGH** SET_PIN_HIGH(PORTC, 5)
- #define **MOTOR2_1_LOW** SET_PIN_LOW(PORTC, 5)

Functions

- void **initBoard** (void)

4.3.1 Detailed Description

Initializes board.

This function was prewritten for the project and hasn't been altered.

Author

Tom Neutens

Date

Jan 19, 2016

4.4 dwenguinoIO.c File Reference

```
#include "../HeaderFiles/dwenguinoIO.h"
```

Functions

- void initLED ()

  Initializes the LEDs by setting the registers right.
- void initButtons ()

  Initializes the buttons by setting the registers right.

4.4.1 Detailed Description

Author

Antoine, Sebastiaan

4.4.2 Function Documentation

4.4.2.1 initButtons()

```
void initButtons ( )
```

Initializes the buttons by setting the registers right.

Returns

Void

4.4.2.2 initLED()

```
void initLED ( )
```

Initializes the LEDs by setting the registers right.

Returns

Void

4.5 dwenguinoIO.h File Reference

This file contains the initialization procedures for the buttons and the LEDs.

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

Functions

- void initLED ()

  Initializes the LEDs by setting the registers right.
- void initButtons ()

  Initializes the buttons by setting the registers right.

4.5.1 Detailed Description

This file contains the initialization procedures for the buttons and the LEDs.

Author

Antoine, Sebastiaan

4.5.2 Function Documentation

4.5.2.1 initButtons()

```
void initButtons ( )
```

Initializes the buttons by setting the registers right.

Returns

Void

4.5.2.2 initLED()

```
void initLED ( )
```

Initializes the LEDs by setting the registers right.

Returns

Void

4.6 dwenguinoLCD.h File Reference

Enables basic print to LCD functionality.

```
#include "dwenguinoBoard.h"
#include <avr/delay.h>
#include "dwenguinoAccelerometer.h"
```

Data Structures

- struct lcd_info_type

Macros

- #define **LCD_WIDTH** 16
- #define **LCD_HEIGHT** 2
- #define **LCD_LASTLINE** (LCD_HEIGHT - 1)
- #define **LCD_LASTPOS** (LCD_WIDTH - 1)
- #define **backlightOn**() (LCD_BACKLIGHT_ON)
- #define **backlightOff**() (LCD_BACKLIGHT_OFF)
- #define **appendStringToLCD**(message) appendStringToLCD_((const char∗)(message))

Functions

- void **initLCD** (void)
- void **clearLCD** (void)
- void **commandLCD** (const BYTE c)
- void **setCursorLCD** (BYTE l, BYTE p)
- void **appendCharToLCD** (const char c)
- void **printCharToLCD** (const char s, BYTE l, BYTE p)
- void **appendIntToLCD** (int i)
- void **printIntToLCD** (int i, BYTE l, BYTE p)
- void **appendStringToLCD_** (const char ∗message)
- void **appendStringToLCDcharptr** (char ∗message)
- void printStringToLCD (char ∗message, BYTE l, BYTE p)

  Altered function. Didn't work properly before.
- void appendUintToLCD (unsigned int i)

  Added function. Similar to appendIntToLCD(int i)
- void printUintToLCD (unsigned int i, BYTE l, BYTE p)

  Added function. Similar to printIntToLCD(int i, BYTE l, BYTE p)
- void dataLCD (const BYTE c)

  Added function. Adds bit-strings to LCD RAM.

Variables

- struct lcd_info_type **lcd_info**

4.6.1 Detailed Description

Enables basic print to LCD functionality.

This function was prewritten for the project and has been slightly altered. The altered/added functions are annotated.

Author

Tom Neutens

Date

Jan 19, 2016

4.6.2 Function Documentation

4.6.2.1 appendUintToLCD()

```
void appendUintToLCD (
             unsigned int i )
```

Added function. Similar to *appendIntToLCD(int i)*

Parameters

i	Unsigned int to append.

Returns

Void

4.6.2.2 dataLCD()

```
void dataLCD (
             const BYTE c )
```

Added function. Adds bit-strings to LCD RAM.

Parameters

c	5-bit bit-string, this is one line of a 8x5 character.

Returns

Void

4.6.2.3 printStringToLCD()

```
void printStringToLCD (
            char * message,
            BYTE l,
            BYTE p )
```

Altered function. Didn't work properly before.

Parameters

message	String to print.
l	Row.
p	Column.

Returns

Void

4.6.2.4 printUintToLCD()

```
void printUintToLCD (
            unsigned int i,
            BYTE l,
            BYTE p )
```

Added function. Similar to *printIntToLCD(int i, BYTE l, BYTE p)*

Parameters

i	Unsigned int to print.
l	Row.
p	Column.

Returns

Void

4.7 dwenguinoLumberjack.c File Reference

```
#include "../HeaderFiles/dwenguinoLumberjack.h"
```

Functions

- Game ∗ InitGame ()

  Allocates space to the Game struct and the tree array and assigns the right values to the variables of the Game struct.
- void DeleteGame (Game ∗game)

  Frees up the space allocated to the game.
- void UpdateTree (Game ∗game)

  Updates the branches of the tree.
- void CheckGameOver (Game ∗game)

  Verifies the state of the LEDs and the state of the game for a "Game Over" and changes the Game struct accordingly.
- void CheckCollision (Game ∗game)

  Verifies if a branch is colliding with Jack and changes the game state and LEDs accordingly.
- void UpdateTilt (Game ∗game)

  Changes the tilt variable depending on the pitch of the accelerometer.
- void UpdateSide (Game ∗game)

  Updates the side on which Jack is standing, depending on the roll angle of the accelerometer.
- void setDifficulty (Game ∗game, int difficulty)

  Sets the difficulty of the game.

4.7.1 Detailed Description

Author

Antoine, Sebastiaan

4.7.2 Function Documentation

4.7.2.1 CheckCollision()

```
void CheckCollision (
            Game * game )
```

Verifies if a branch is colliding with Jack and changes the game *state* and LEDs accordingly.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.7.2.2 CheckGameOver()

```
void CheckGameOver (
            Game * game )
```

Verifies the state of the LEDs and the *state* of the game for a "Game Over" and changes the *Game* struct accordingly.

Returns

Void

In case of a "Game-Over", the tree array is emptied, the LEDs are all turned off and game state is set to *DEAD*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.7.2.3 DeleteGame()

```
void DeleteGame (
            Game * game )
```

Frees up the space allocated to the game.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.7.2.4 InitGame()

```
Game* InitGame ( )
```

Allocates space to the *Game* struct and the tree array and assigns the right values to the variables of the *Game* struct.

This function needs to be called before using any of the functions in this file.

Returns

Pointer to the address of the *Game* struct.

4.7.2.5 setDifficulty()

```
void setDifficulty (
            Game * game,
            int difficulty )
```

Sets the *difficulty* of the game.

Parameters

game	Pointer to the *Game* struct.
difficulty	New difficulty to be set.

Returns

Void

4.7.2.6 UpdateSide()

```
void UpdateSide (
            Game * game )
```

Updates the side on which Jack is standing, depending on the roll angle of the accelerometer.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.7.2.7 UpdateTilt()

```
void UpdateTilt (
            Game * game )
```

Changes the *tilt* variable depending on the pitch of the accelerometer.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.7.2.8 UpdateTree()

```
void UpdateTree (
            Game * game )
```

Updates the branches of the tree.

This function generates new branches at the right time and updates the existing branches to fall one position.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8 dwenguinoLumberjack.h File Reference

This file contains all the functions that are made specifically for the game.

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

Data Structures

- struct Game

  Struct containing all game variables.

Macros

- #define **PITCH_POINT** 10
- #define **MENU** 0
- #define **ALIVE** 1
- #define **DEAD** 2

Typedefs

- typedef struct Game **Game**

Functions

- Game ∗ InitGame ()

  Allocates space to the Game struct and the tree array and assigns the right values to the variables of the Game struct.
- void DeleteGame (Game ∗game)

  Frees up the space allocated to the game.
- void UpdateTree (Game ∗game)

  Updates the branches of the tree.
- void UpdateSide (Game ∗game)

  Updates the side on which Jack is standing, depending on the roll angle of the accelerometer.
- void UpdateTilt (Game ∗game)

  Changes the tilt variable depending on the pitch of the accelerometer.
- void setDifficulty (Game ∗game, int difficulty)

  Sets the difficulty of the game.
- void CheckCollision (Game ∗game)

  Verifies if a branch is colliding with Jack and changes the game state and LEDs accordingly.
- void CheckGameOver (Game ∗game)

  Verifies the state of the LEDs and the state of the game for a "Game Over" and changes the Game struct accordingly.

4.8.1 Detailed Description

This file contains all the functions that are made specifically for the game.

Almost all changes happen by changing the variables in the *Game* struct.

Author

Antoine, Sebastiaan

4.8.2 Function Documentation

4.8.2.1 CheckCollision()

```
void CheckCollision (
            Game ∗ game )
```

Verifies if a branch is colliding with Jack and changes the game *state* and LEDs accordingly.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8.2.2 CheckGameOver()

```
void CheckGameOver (
            Game * game )
```

Verifies the state of the LEDs and the *state* of the game for a "Game Over" and changes the *Game* struct accordingly.

Returns

Void

In case of a "Game-Over", the tree array is emptied, the LEDs are all turned off and game state is set to *DEAD*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8.2.3 DeleteGame()

```
void DeleteGame (
            Game * game )
```

Frees up the space allocated to the game.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8.2.4 InitGame()

```
Game* InitGame ( )
```

Allocates space to the *Game* struct and the tree array and assigns the right values to the variables of the *Game* struct.

This function needs to be called before using any of the functions in this file.

Returns

Pointer to the address of the *Game* struct.

4.8.2.5 setDifficulty()

```
void setDifficulty (
             Game * game,
             int difficulty )
```

Sets the *difficulty* of the game.

Parameters

game	Pointer to the *Game* struct.
difficulty	New difficulty to be set.

Returns

Void

4.8.2.6 UpdateSide()

```
void UpdateSide (
             Game * game )
```

Updates the side on which Jack is standing, depending on the roll angle of the accelerometer.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8.2.7 UpdateTilt()

```
void UpdateTilt (
            Game * game )
```

Changes the *tilt* variable depending on the pitch of the accelerometer.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.8.2.8 UpdateTree()

```
void UpdateTree (
            Game * game )
```

Updates the branches of the tree.

This function generates new branches at the right time and updates the existing branches to fall one position.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9 dwenguinoPrint.c File Reference

```
#include "../HeaderFiles/dwenguinoPrint.h"
```

Functions

- void generateLumberjack (Game ∗game)

  Puts the right Jack characters in the LCD RAM.
- void generateBranch (Game ∗game)

  Puts the right branch characters in the LCD RAM.
- void generateSmallBranch (Game ∗game)

  Puts the right small branch characters in the LCD RAM.

- void printGameToLCD (Game ∗game)

  Prints the tree together with Jack to the LCD.
- void printGameOver (Game ∗game)

  Prints "Game Over" screen.
- void printMenu (Game ∗game, int difficulty)

  Prints "Menu" screen.

4.9.1 Detailed Description

Author

Antoine, Sebastiaan

4.9.2 Function Documentation

4.9.2.1 generateBranch()

```
void generateBranch (
             Game ∗ game )
```

Puts the right branch characters in the LCD RAM.

The right branch character depends on the game *counter*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9.2.2 generateLumberjack()

```
void generateLumberjack (
             Game ∗ game )
```

Puts the right Jack characters in the LCD RAM.

The jack characters depends on the game *state* and the *counter* for the animation.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9.2.3 generateSmallBranch()

```
void generateSmallBranch (
            Game * game )
```

Puts the right small branch characters in the LCD RAM.

The right small branch character depends on the *counter* variable.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9.2.4 printGameOver()

```
void printGameOver (
            Game * game )
```

Prints "Game Over" screen.

The "Game Over" screen contains the game *score*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9.2.5 printGameToLCD()

```
void printGameToLCD (
            Game * game )
```

Prints the tree together with Jack to the LCD.

This function is heavily dependent on the characters created in the *generate* functions.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.9.2.6 printMenu()

```
void printMenu (
            Game * game,
            int difficulty )
```

Prints "Menu" screen.

Parameters

game	Pointer to the *Game* struct.
difficulty	The difficulty to print on screen.

Returns

Void

4.10 dwenguinoPrint.h File Reference

This file contains all the functions that generate characters and print them on the LCD.

```
#include "dwenguinoAccelerometer.h"
#include "dwenguinoLCD.h"
```

Macros

- #define **Jack_bottom** 0
- #define **Jack_top** 1
- #define **EMPTY_BRANCH_L** 2
- #define **EMPTY_BRANCH_R** 3
- #define **Branch_L** 4
- #define **Branch_R** 5

Functions

- void generateLumberjack (Game ∗game)

  Puts the right Jack characters in the LCD RAM.
- void generateBranch (Game ∗game)

  Puts the right branch characters in the LCD RAM.
- void generateSmallBranch (Game ∗game)

  Puts the right small branch characters in the LCD RAM.
- void printGameToLCD (Game ∗game)

  Prints the tree together with Jack to the LCD.
- void printGameOver (Game ∗game)

  Prints "Game Over" screen.
- void printMenu (Game ∗game, int difficulty)

  Prints "Menu" screen.

4.10.1 Detailed Description

This file contains all the functions that generate characters and print them on the LCD.

Author

Antoine, Sebastiaan

4.10.2 Function Documentation

4.10.2.1 generateBranch()

```
void generateBranch (
             Game ∗ game )
```

Puts the right branch characters in the LCD RAM.

The right branch character depends on the game *counter*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.10.2.2 generateLumberjack()

```
void generateLumberjack (
            Game * game )
```

Puts the right Jack characters in the LCD RAM.

The jack characters depends on the game *state* and the *counter* for the animation.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.10.2.3 generateSmallBranch()

```
void generateSmallBranch (
            Game * game )
```

Puts the right small branch characters in the LCD RAM.

The right small branch character depends on the *counter* variable.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.10.2.4 printGameOver()

```
void printGameOver (
            Game * game )
```

Prints "Game Over" screen.

The "Game Over" screen contains the game *score*.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.10.2.5 printGameToLCD()

```
void printGameToLCD (
            Game * game )
```

Prints the tree together with Jack to the LCD.

This function is heavily dependent on the characters created in the *generate* functions.

Parameters

game	Pointer to the *Game* struct.

Returns

Void

4.10.2.6 printMenu()

```
void printMenu (
            Game * game,
            int difficulty )
```

Prints "Menu" screen.

Parameters

game	Pointer to the *Game* struct.
difficulty	The difficulty to print on screen.

Returns

Void

4.11 main.c File Reference

Enables button interrupts and ties everything together for the game to work as intended.

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <math.h>
#include "HeaderFiles/dwenguinoBoard.h"
#include "HeaderFiles/dwenguinoLCD.h"
#include "HeaderFiles/dwenguinoAccelerometer.h"
#include "HeaderFiles/dwenguinoPrint.h"
#include "HeaderFiles/dwenguinoLumberjack.h"
#include "HeaderFiles/dwenguinoIO.h"
```

Functions

- ISR (INT4_vect)

  Interrupt function for the west button. Decreases the difficulty.
- ISR (INT5_vect)

  Interrupt function for the south button. Starts the game.
- ISR (INT6_vect)

  Interrupt function for the east button. Increases the difficulty.
- int main ()

  Initializes everything and starts the game-loop.

Variables

- int **status** = MENU
- int **difficulty** = 1
- int **buttonwait** = 0

4.11.1 Detailed Description

Enables button interrupts and ties everything together for the game to work as intended.

Author

Antoine, Sebastiaan

4.11.2 Function Documentation

4.11.2.1 main()

```
int main ( )
```

Initializes everything and starts the game-loop.

Returns

Should not return.

Index