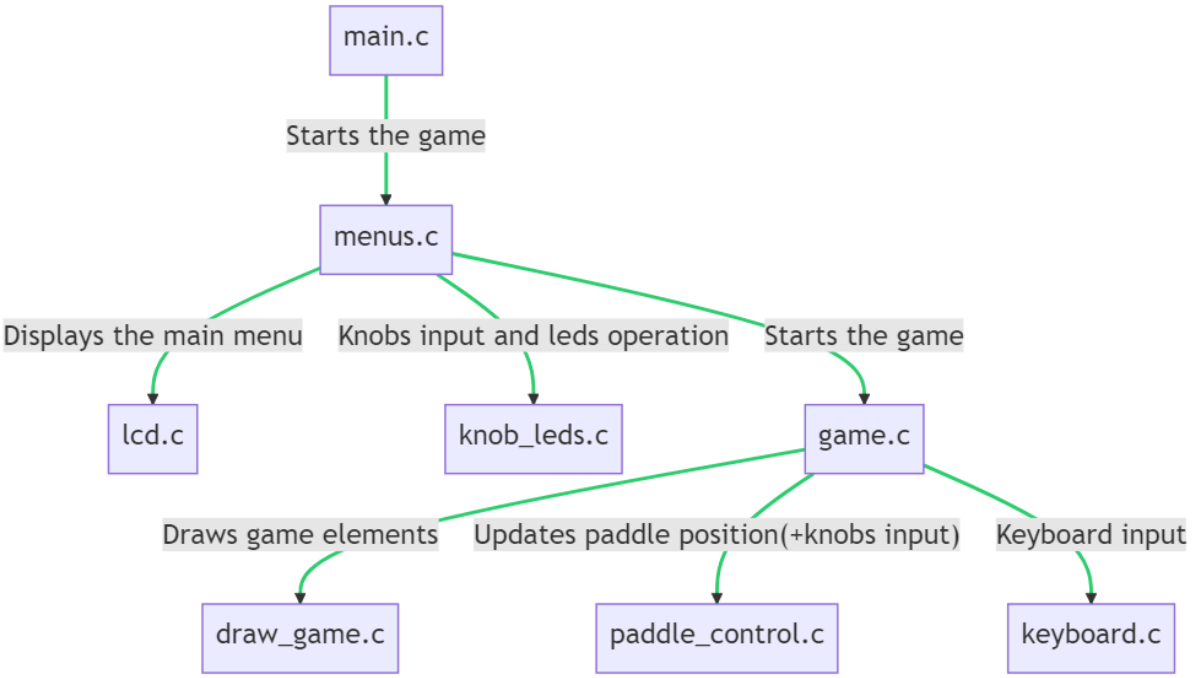
Breakout APO

Made by:EIDENIS KASPERAVICIUS
RAMAZAN YESTEKBAYEV

Application architecture



Block diagram of game architecture:

- The "main.c" file starts the game and interacts with the "menus.c" file.
- The "menus.c" file displays the main menu on the "lcd.c" file and gets user input from the "knob_leds.c" file.
- The "menus.c" file also starts the game in the "game.c" file.
- The "game.c" file draws game elements using the "draw_game.c" file, updates the paddle position using the "paddle_control.c" file, and gets user input from the "keyboard.c" file.

Functions and global variables

Here are the detailed explanations of the function arguments and return values:

main.h & main.c
- **'main(int argc, char *argv[])'**: This is the entry point of the program. It takes two arguments: 'argc' which is the count of command-line arguments, and 'argv' which is an array of command-line arguments. It returns an integer, which is the exit status of the program.

game.h & game.c
- **'play_game(menu_item_t difficulty_menu_item, int *score)'**: This function starts the game. It takes two arguments: **'difficulty_menu_item'** which is the selected difficulty level, and 'score' which is a pointer to an integer to store the score. It returns an integer indicating whether the game was won (1) or lost (0).

keyboard.h & keyboard.c
- **'init_stdin()'**: This function initializes the standard input for non-blocking input. It does not take any arguments and does not return anything.
- **'get_char()'**: This function reads a character from the standard input without blocking. It does not take any arguments and returns the character read.

menus.h & menus.c
- **'display_main_menu()'**: This function displays the main menu and returns the selected menu item. It does not take any arguments.
- **'display_final_screen(int score, int won)'**: This function displays the final screen after the game ends. It takes two arguments: **'score'** which is the final score, and 'won' which is a boolean indicating whether the game was won. It does not return anything.

draw_game.h & draw_game.c
- **'draw_paddle(Paddle *paddle, uint16_t color)'**: This function draws the paddle on the screen. It takes two arguments: **'paddle'** which is a pointer to the paddle, and **'color'** which is the color to draw the paddle. It does not return anything.
- **'draw_ball(Ball *ball, uint16_t color)'**: This function draws the ball on the screen. It takes two arguments: **'ball'** which is a pointer to the ball, and **'color'** which is the color to draw the ball. It does not return anything.
- **'draw_brick(Brick *brick, uint16_t color)'**: This function draws a brick on the screen. It takes two arguments: **'brick'** which is a pointer to the brick, and **'color'** which is the color to draw the brick. It does not return anything.

paddle_control.h & paddle_control.c
- **'update_paddle_pos(Paddle *paddle, int knob_val)'**: This function updates the position of the paddle based on the knob value. It takes two arguments: **'paddle'** which is a pointer to the paddle, and **'knob_val'** which is the value of the knob. It does not return anything.

knob_leds.h & knob_leds.c

- **'init_knob_leds()'**: This function initializes the knob and LEDs. It does not take any arguments and returns an integer. If the initialization is successful, it returns 0, otherwise -1.
- **show_lives(int lives)**: This function displays the number of lives on the LEDs. It takes one argument: lives which is the number of lives to be displayed.
- **get_knob(int *moved, int *pressed)**: This function gets the current state of the knob. It takes two arguments: moved which is a pointer to an integer to store whether the knob was moved, and pressed which is a pointer to an integer to store whether the knob was pressed.

lcd.h & lcd.c

- **init_lcd()**: This function initializes the LCD. It does not take any arguments and returns an integer. If the initialization is successful, it returns 0, otherwise -1.
- **fb_clear()**: This function clears the framebuffer. It does not take any arguments and does not return anything.
- **fb_draw_pixel(int x, int y, int c)**: This function draws a pixel at the specified coordinates in the framebuffer. It takes three arguments: x and y which are the coordinates of the pixel, and c which is the color of the pixel.
- **fb_update()**: This function updates the LCD with the contents of the framebuffer. It does not take any arguments and does not return anything.

Global Variables

- In **knob_leds.c**, the global variables are **LED_PRESETS[]**, **spiled_base**, and **last_knob_val**.
- In **lcd.c**, the global variables are **fb_pixels[]** and **parlcd_base**.
- In **menus.c**, the global variables are **menu_items[]** and **menu_items_count**.

Global Defines (types.h)

Constant	Value	Purpose
FB_WIDTH	480	Frame buffer width (= display width) in pixels
FB_HEIGHT	320	Frame buffer height (= display height) in pixels
PADDLE_WIDTH	80	Paddle width in pixels
PADDLE_HEIGHT	10	Paddle height in pixels
BALL_SIZE	10	Ball diameter in pixels
BRICK_ROWS	5	Number of brick rows
BRICK_COLS	10	Number of brick columns
BRICK_WIDTH	40	Brick row in pixels
BRICK_HEIGHT	20	Brick height in pixels

BALL_SPEED	6.0	Ball speed in pixels per frame

Used types (types.h)

typedef enum { MENU_EASY, MENU_NORMAL, MENU_HARD, MENU_EXIT, } menu_item_t;	Items shown on main menu
typedef struct { int x; int y; int width; int height; } Paddle;	Paddle • x - horizontal coordinate • y - vertical coordinate • width - paddle width, pixels • height - paddle height, pixels
typedef struct { int x; int y; int size; int dx; int dy; } Ball;	Ball • x - horizontal coordinate • y - vertical coordinate • size - diameter in pixels • dx - horizontal speed, pixels per frame • dy - vertical speed, pixels per frame
typedef struct { int x; int y; int width; int height; int active; int points; uint16_t color; } Brick;	Brick • x - horizontal coordinate • y - vertical coordinate • width - brick width, pixels • height - brick height, pixels • active - 0 if brick was destroyed and should not be drawn; 1 otherwise • points - 0 to 3; how many points player gains after destroying brick • color - RGB565 color code

User Manual

1. **Starting the Game**: Connect to your MicroZed board using serial port. Use the 'ip addr show' command to find the IP address of your board. Check the Makefile and change the IP address to match your board's IP address, then run 'make run'. Alternatively, run the command 'make run TARGET_IP=192.168.xxx.xxx' to start the game. Dependencies is already set up in default MAKEFILE, just change the IP address. Serial connection, parameters:

- Device: /dev/ttyUSB0
- Baudrate: 115200
- No parity and single stop bit
- Login: root
- The password for local serial port login will be provided by lab tutor

There are many utilities providing access to serial port communication, for example gtkterm or even direct command line utility tio -b 115200 /dev/ttyUSB0 from terminal. After you obtain the IP address of the board it is much more comfortable to use SSH for access. If you have a Microsoft Windows only computer available, you can use PuTTY program to access the serial port and connect to the board through SSH.

Note, SSH, MAKE utility and gcc cross-compiler(arm-linux-gnueabihf-gdb) (apt install crossbuild-essential-armhf) should be installed on your system.

2. **Main Menu**: The main menu will be displayed on the screen. The menu items include "Easy", "Normal", "Hard", and "Exit Game".

3. **Selecting Difficulty**: Use the blue blue knob to navigate through the menu items. Press the blue blue knob to select the highlighted menu item. The difficulty levels include "Easy", "Normal", and "Hard". The difficulty level determines the number of lives: "Easy" gives you 3 lives, "Normal" gives you 2 lives, and "Hard" gives you 1 life.

4. **Playing the Game**: Once a difficulty level is selected, the game starts. Control the paddle using the blue knob. The paddle moves left or right based on the blue knob's movement.

5. **Scoring**: Points are scored by hitting bricks with the ball. The score is displayed on the LEDs.

6. **Lives**: The number of lives is displayed on the LEDs. You lose a life when the ball hits the bottom of the screen.

7. **Ball Direction**: If the ball collides with the left side of the paddle, it bounces off at a left angle. If it hits the center of the paddle, it bounces straight up. If it hits the right side of the paddle, it bounces off at a right angle.

8. **Ending the Game**: The game ends when all bricks are destroyed (win) or when all lives are lost (lose). The final score is displayed on the screen.

9. **Exiting the Game**: To exit the game, select "Exit Game" from the main menu.

Additional information

- Working with MicroZed:
  cw.fel.cvut.cz/wiki/courses/b35apo/en/documentation/mz_apo-howto/start

- The Makefile is a crucial component of the build process for the Breakout game. It is a script used by the 'make' build automation tool to compile and link the game. Here are some key points about the Makefile:

  1. **Compiler**: The Makefile specifies the compiler to be used. By default, it is set to use the GCC (GNU Compiler Collection) for compiling the C source files.

  2. **Compiler Flags**: The Makefile also specifies compiler flags, which control various aspects of compilation, such as optimization level, warning level, and the inclusion of debug information.

  3. **Dependencies**: The Makefile lists the dependencies for each object file. This means that if any of the source files listed as a dependency changes, the object file will be recompiled.

  4. **Build Targets**: The Makefile defines several build targets, such as 'all' (to build everything), 'clean' (to remove all compiled files), and 'run' (to run the game). These targets can be executed by running 'make <target>'.

  5. **IP Address**: The Makefile includes a variable for the target IP address. This is used when running the game on the MicroZed board. You can set this variable in the Makefile, or you can specify it when running 'make run' by using 'make run TARGET_IP=<ipaddr>'.

  6. **SSH** and **SCP**: The Makefile uses SSH (Secure Shell) to connect to the MicroZed board and SCP (Secure Copy) to transfer the compiled game to the board.

Remember to check the Makefile and adjust any settings as necessary for your specific setup and requirements.
Additional information about make utility: gnu.org/software/make/