Project report: the implementation Arkanoid like game on STM32f4 board

EL6483 Real Time Embedded System
David Paul Pena (N14052208, dpp289) Shuaiyu Liang (N14501460, sl5352)

I. Abstract

Arkanoid games usually require the player to control the paddle movement and bounce the ball against the bricks to clear up bricks. We notice this process involves with reading data from 3-axis accelerometer, using the LCD screen display, and user interaction with the touch screen. Inspired by those ideas, we implemented an Arkanoid-like game on the STM32f4 board with a SSD1289 touch screen. The game is running in a real time manner.

II. Introduction

- **Game logic overall**

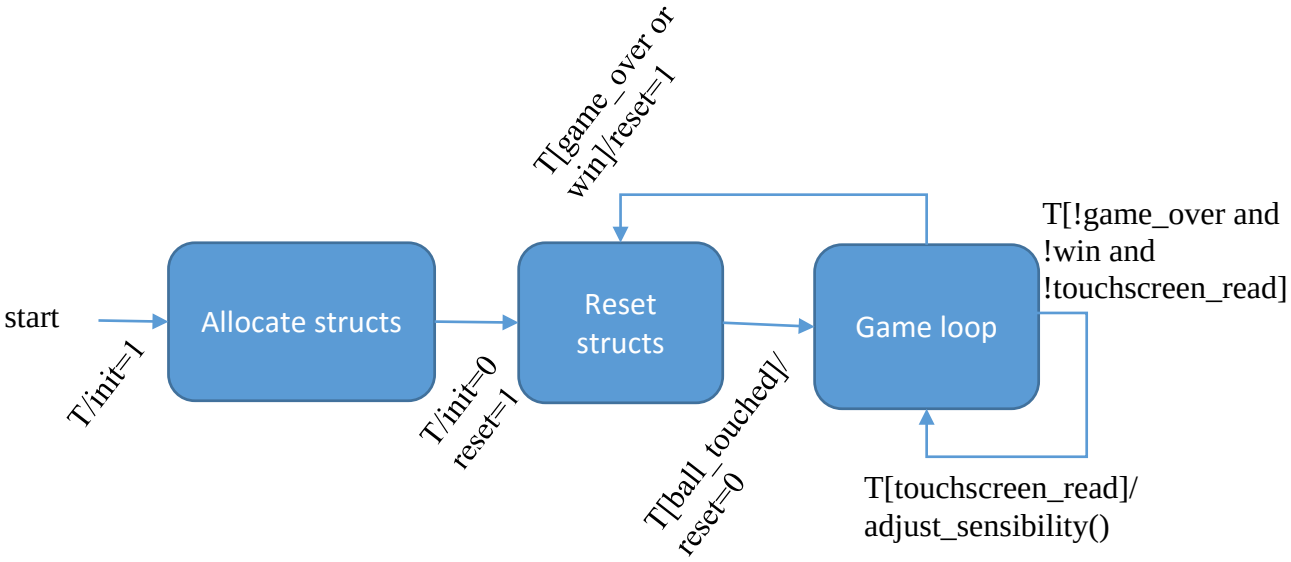
The game logic is pretty intuitive. The parameters are defined below. During the game we will keep monitoring and manipulating those parameters. The information is updated every 50ms, which is 20 frame per second. Thus on the one hand, we will not be experiencing any discontinuity on display. On the other hand, the communication and computation should not be too heavy to introduce unpleasant latency.

Table 1. Objects and their parameters

Objects	Parameters
Ball	Position, radius, speed
Paddle	Position, length, height
Walls	Limits
Array of bricks	Position, length, height, active flag
Sensitivity	Sensitivity level
Score	Score

Basically, we need to define the range of the limit of the screen. Every element drawn on the screen needs its corresponding central position. While starting the game, we need to define the velocity of the ball, which will be based on whether the ball has a collision with bricks, or walls, or the paddle, or simply dropping out of paddle and collide with the bottom of the screen. Based on these three kinds of collision, we defined three functions to check them. And the core idea is that the ball will reverse its speed to from the original speed if collide. Another module is paddle moving control. We use 3-axis accelerometer to read the pitch information of the board and control the paddle moving at different speed. A timed_task is created and called at the main function every 50ms. The draw game function is created for drawing everything on the LCD screen. To put all functionalities together, a function called update_game is created. Basically this is the one to plug in the timed_task.

- **Game state machine**



III. Implementation

- **Hardware components**

  The game is implemented on STM32f4 board with a SSD1289 touch screen. The game also uses the 3-axis accelerometers on STM32f4.

- **Interfaces**

  The LCD module is connected to the STM32f4 board in parallel mode using the Flexible Static Memory Controller of the STM32f4. The touchscreen module is connected using SPI. These two modules use the following GPIO pins of the STM32f4 board:

LCD pin name	STM32F4 pin name
D0	PD14
D1	PD15
D2	PD0
D3	PD1
D4	PE7
D5	PE8
D6	PE9
D7	PE10
D8	PE11
D9	PE12
D10	PE13

LCD pin name	STM32F4 pin name
D11	PE14
D12	PE15
D13	PD8
D14	PD9
D15	PD10
CS	PD7 (NE1)
RS	PD11 (FSMC-A16)
RD	PD4 (NOE)
WR	PD5 (NWE)
RESET	+3V(1K Pull up) or GPIO
TP_IRQ	PD6
TP_SCK	PB13
TP_SI	PB15
TP_SO	PB14
TP_CS	PB12
BL_CNTL	PA10

- **Code structure**

  - Main function

To start the system, all parameters are initialized first, LCD is initialized, and accelerometer is initialized. Then add the update_game into the timed_task and set the calling time interval to 0.05 second. In the while(1) loop, call the update function to update the game.

  - Game logic set

In the game logic folder, we mainly created the following files:

Table 2. game logic set files

File name	File function
collisions.c	Functions to detect collisions of the ball the the game limits, paddle o bricks.
draw.c	Functions related to drawing and erasing the game elements on the LCD
game.c	Game loop
init.c	Functions to allocate of the elements of the game in the HEAP
paddle_movement.c	Update the paddle position using the accelerometers
timed_tasks.c	Implementation of timed tasks. Definition of the update() function that checks if it has to call a timed task for its execution

touch_management.c	Check and change parameters involving the touch screen

In the *collision.c*, the collision to paddle, game limits, and bricks are checked. If the ball collide with the paddle, which means when ball is moving, ball's y value add its radius and y speed goes beyond the paddle's y value + paddle height, and ball's x values within the paddle range, the ball get a reversed y velocity and remains its original x velocity. The ball acts following the same logic when colliding with the game limits and with bricks. When colliding with bricks, it changes the ball velocity as well as changes the collided bricks active flag and erase it. It also updates the score and draw it immediately. Another function in this file checks the eliminated bricks number and returns end flag if all bricks are eliminated and the game is ended.

In the *draw.c*, the ball and paddle are erased and drawed together because these elements will change their position on every frame. To draw something on the LCD basically just set the drawing color to green and to erase something than set the drawing color to black. Separately we defined draw ball and paddle, draw score, draw sensibility, erase brick, and put them together draw all the game. Draw all the game is used at the initial phase. Only in this phase we draw all the bricks and each time collision happens, the game erase the corresponding brick. This allows to do not to draw the whole game on each frame. This mitigate some communication burden as well as avoid the screen flip while running the game.

In the *game.c*, we defined an update_game function. Basically it will navigate the program into 3 different states and combine the update data functions with the LCD drawing functions. The initial state, waiting_to_start, and the game running states. Each state has it corresponding actions. If initialize flag set, the game is initialized. If the game is waiting for start, then it will stay in that state until check_start_game returns 1 and move to gaming state. In the gaming state, it will update the game until game_over flag is set or eliminate all the bricks. Every iteration clear last time paddle and ball, then update position and draw them again.

In the *init.c*, basically initialize the variables, set the value as in it in the *structure.h*. Allocate memory in heap to paddle, the ball and the bricks. Also it defines wait function.

In the *paddle_movement.c*, the current accelerometer data is read and stored in a circular buffer. Then uses four past values and one current value to compute the filtered accelerometer value and then compute the pitch. Finally, it moves the paddle as defined by incrementing paddle position, checking that it is inside the game limits.

In the *timed_task.c* defined a timed_task struct, add timed_task function and the update function used in main function.

In the *touch_management.c*, basically read touch position from touch screen, defines a start game condition, and a function to check and update sensibility control. Then the sensibility changes are plugged in to paddle movement speed.

- Headers

  Header files are corresponding to each c file. *structure.h* contains some defined values for the game. The screen limit is set to from 0 to 240 at the narrow edge and 320 to 30 at the wider edge. Bricks are set to 30x14 and totally 18 of them. The paddle is set to 60x10 Ball radius is set at the initial function to 3. These three structs are defined as:

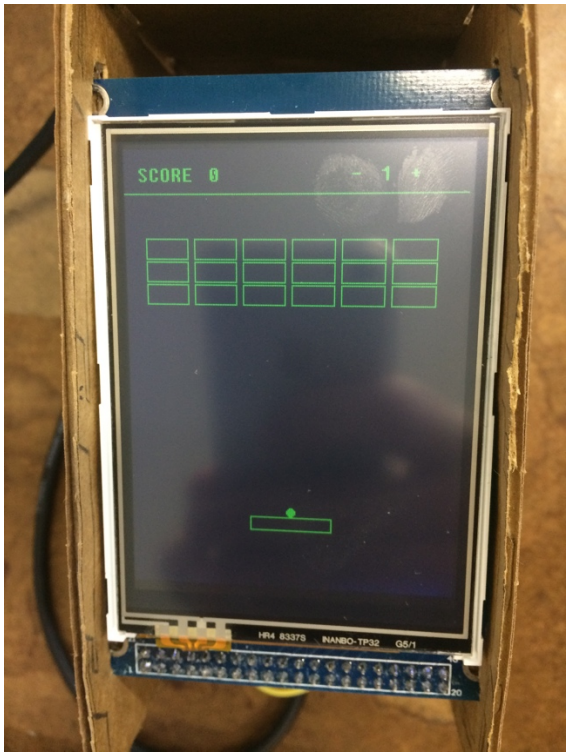
```
typedef struct {
        uint8_t active;
        uint32_t x;
        uint32_t y;
        uint32_t length;
        uint32_t height;
} brick_t;
```

```
typedef struct {
        uint32_t x;
        uint32_t y;
        uint32_t length;
        uint32_t height;
} paddle_t;
```

```
typedef struct {
        uint32_t x;
        uint32_t y;
        int32_t vx;
        int32_t vy;
        uint32_t radius;
} ball_t;
```

IV. Test and analysis

- Displayed game



- Testing the game

  To start the game, tap the region around the paddle. The player can also adjust the sensitivity level before or after starting of the game. After tap the paddle, the ball flies out, and collide with bricks. The score goes up. After eliminate all the bricks, it displays win and reset the game. If the ball drops out of the paddle region the game is reset.

Screen works well, with no latency. Again because we only draw bricks once and erase every collision happens, drawing element in each period is actually only the paddle, the score, the ball and the sensibility. This avoid the flip of screen image.

V. Possible enhancement

The game can be improved by changing the angle of the ball depending on the relative position of the ball respect to the paddle when they collide. This could be done by varying the x velocity and y velocity, but keeping the overall velocity constant.