Pokemon Breaker

Rui Chen, rc3205
Dajing Xu, dx2178
Shao-Fu Wu, sw3385
Bingyao Shi, bs3119

Content

System Architecture

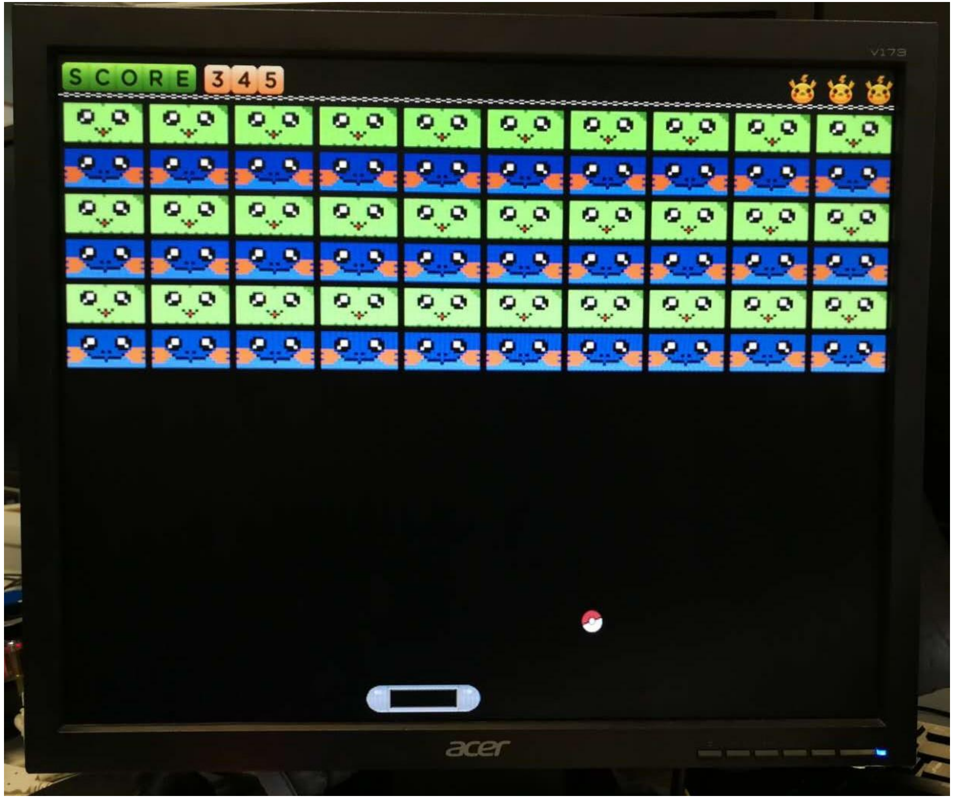
Hardware

- Graphic Display
- Audio Sound

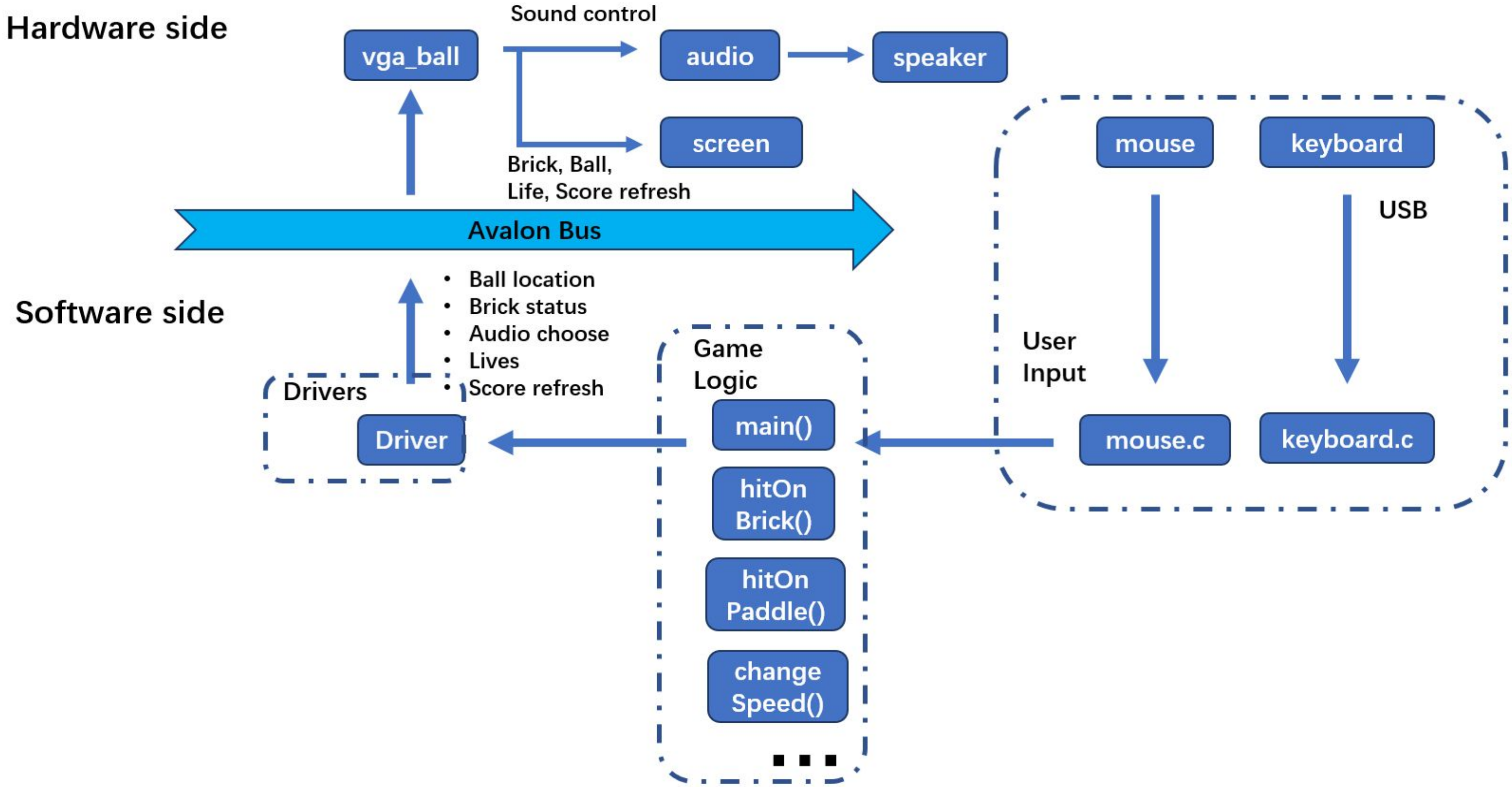
Software

- Inputs
- Game logic

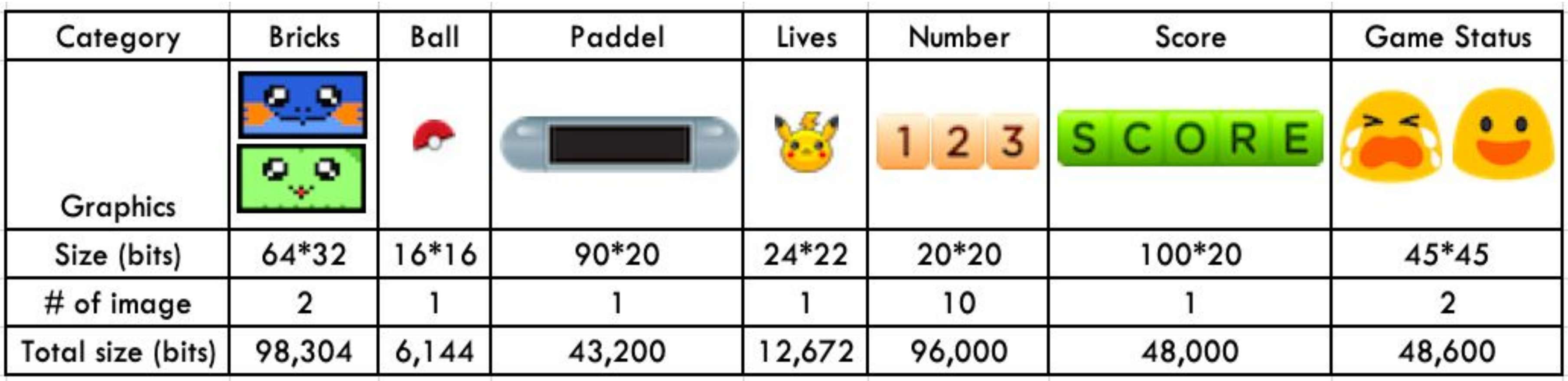
Take a Glance



Architecture



Graphic Display - Memory Budget

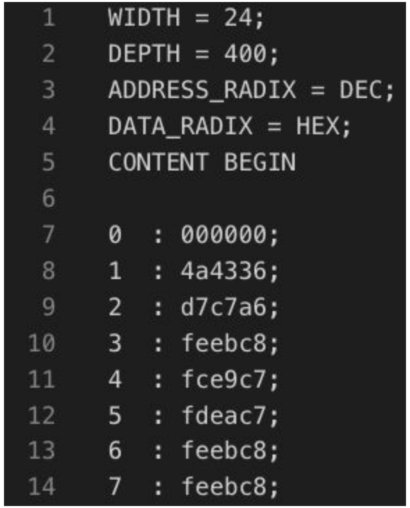


Category	Bricks	Ball	Paddel	Lives	Number	Score	Game Status
Graphics					1 2 3	S C O R E	
Size (bits)	64*32	16*16	90*20	24*22	20*20	100*20	45*45
# of image	2	1	1	1	10	1	2
Total size (bits)	98,304	6,144	43,200	12,672	96,000	48,000	48,600

353 Kbits used out of 4,450 Kbits of embedded memory

Graphic Display - Processing

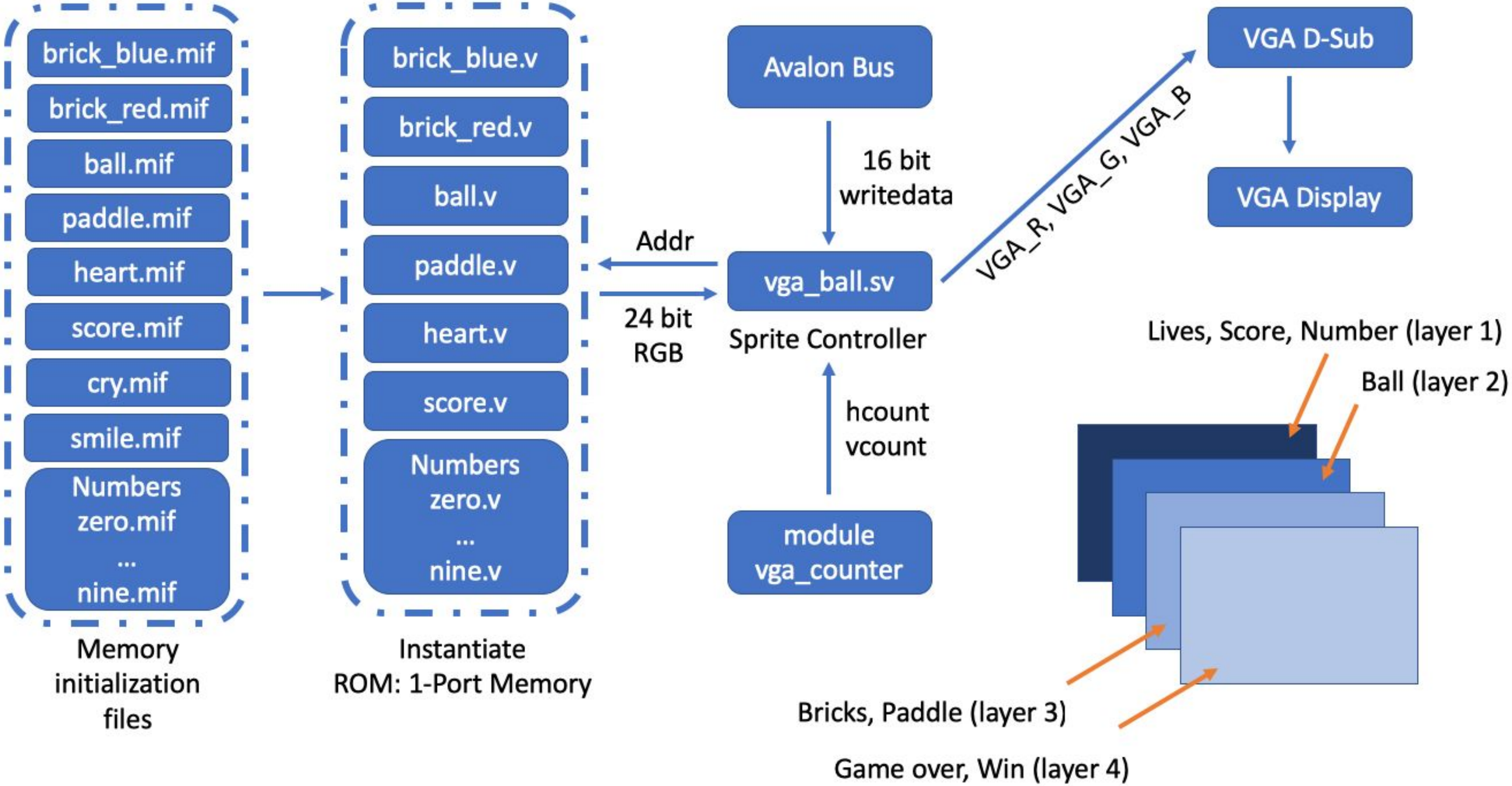
- Use Matlab code to preprocess .png images into .mif files
- Use MegaWizard to configure single-port ROM memory blocks for every sprite
- .mif files contain 24-bit color information for each pixel, 8 bits each for R, G & B



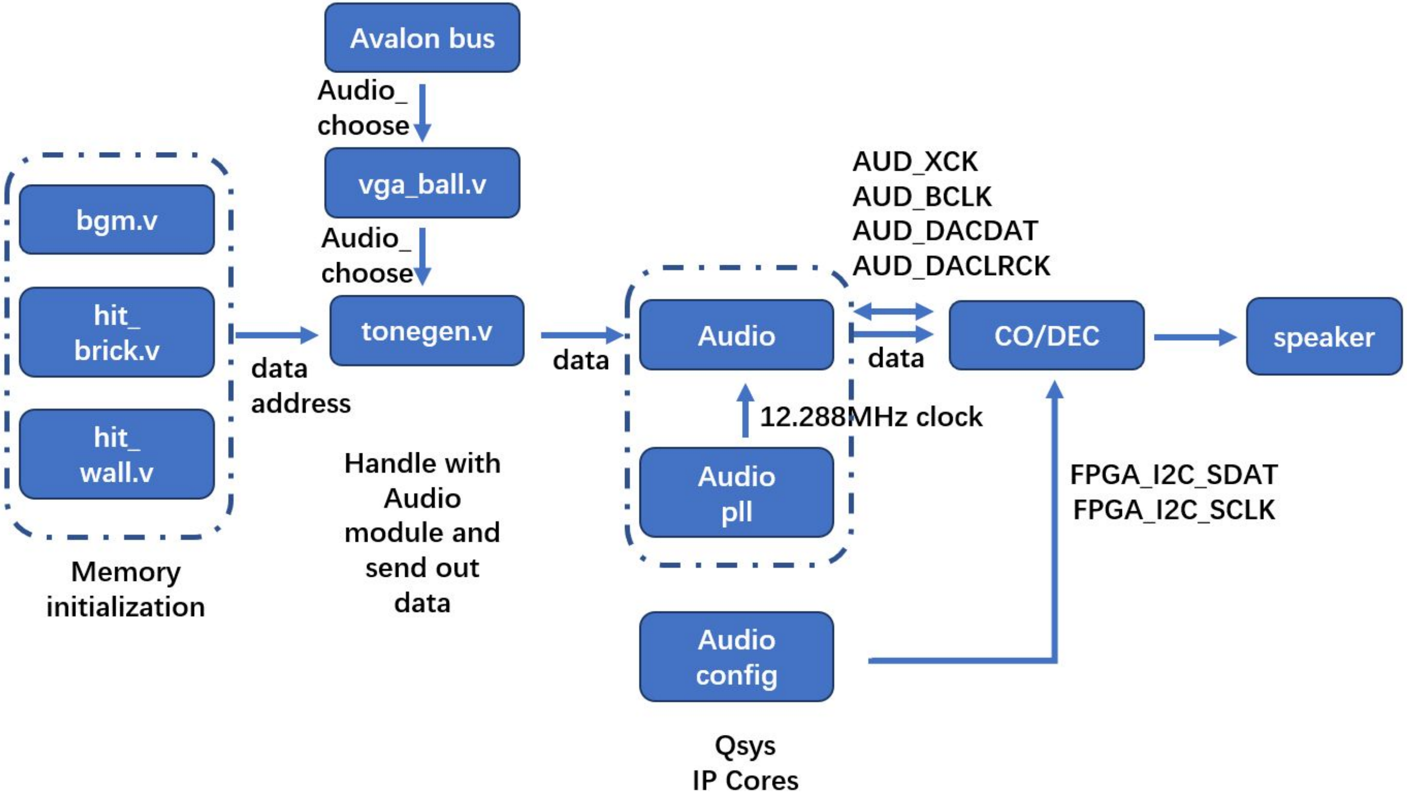
```
WIDTH = 24;
DEPTH = 400;
ADDRESS_RADIX = DEC;
DATA_RADIX = HEX;
CONTENT BEGIN

0   : 000000;
1   : 4a4336;
2   : d7c7a6;
3   : feebc8;
4   : fce9c7;
5   : fdeac7;
6   : feebc8;
7   : feebc8;
```

Graphic Display - Architecture & Layers



Audio Architecture

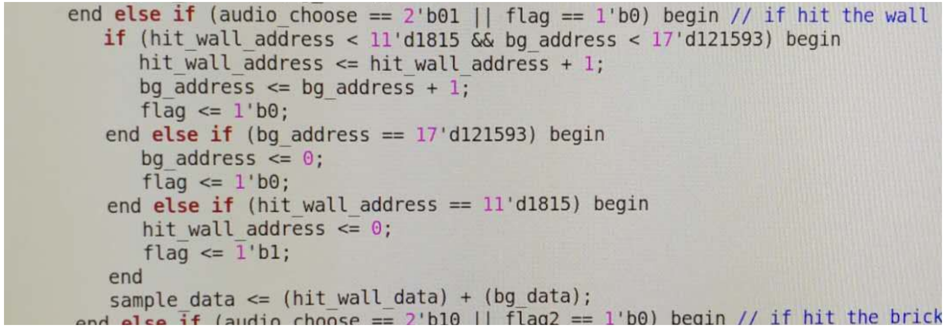


Audio Effect Design

When ball hits on brick or wall, only in a single loop the audio_choose is set, in the next loop it goes back to 0. In our design, a loop is roughly 1.2ms, which is much shorter than the sound effects which are around 0.3s.

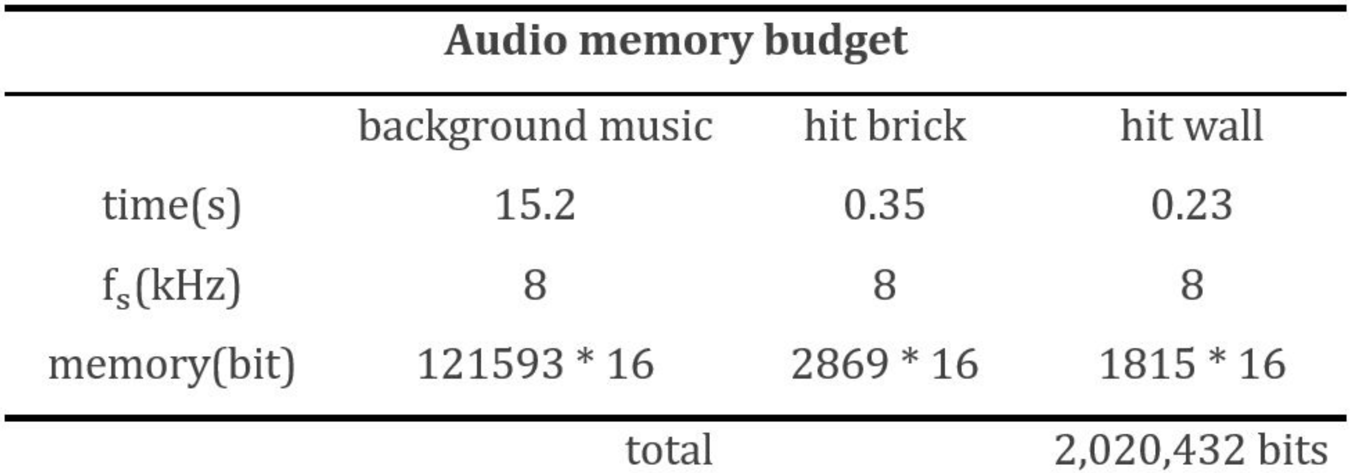
Solve this from hardware side:

Use a flag to mark whether the sound effect is over.



```
end else if (audio_choose == 2'b01 || flag == 1'b0) begin // if hit the wall
    if (hit_wall_address < 11'd1815 && bg_address < 17'd121593) begin
        hit_wall_address <= hit_wall_address + 1;
        bg_address <= bg_address + 1;
        flag <= 1'b0;
    end else if (bg_address == 17'd121593) begin
        bg_address <= 0;
        flag <= 1'b0;
    end else if (hit_wall_address == 11'd1815) begin
        hit_wall_address <= 0;
        flag <= 1'b1;
    end
    sample_data <= (hit_wall_data) + (bg_data);
end else if (audio_choose == 2'b10 || flag2 == 1'b0) begin // if hit the brick
```

Audio Effects Memory Budget



Audio memory budget

	background music	hit brick	hit wall
time(s)	15.2	0.35	0.23
f_s(kHz)	8	8	8
memory(bit)	121593 * 16	2869 * 16	1815 * 16
	total		2,020,432 bits

Audio Effects Summary

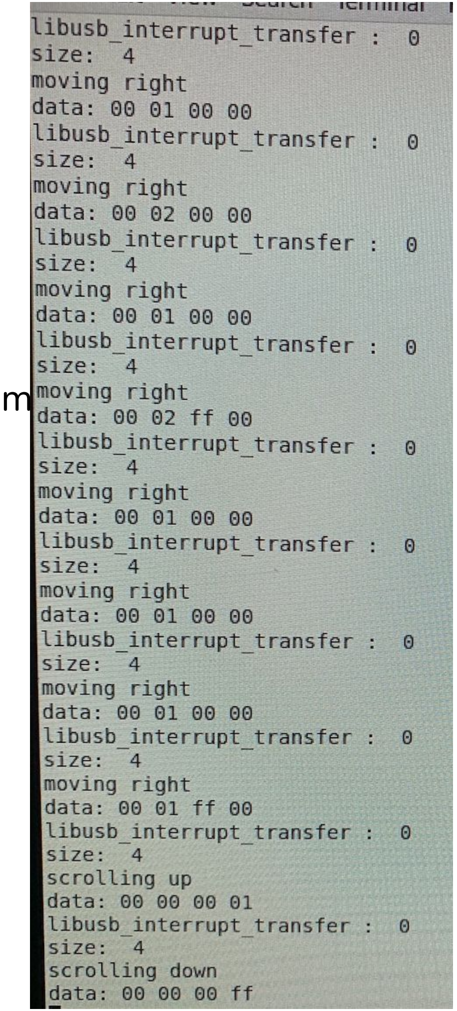
1. Audio effects includes sound effect of hitting on wall and on bricks, sampling rate is 8kHz.
2. Controlled by a audio_choose signal, sending from user space.
3. Audio effects don’t disturb the background music, just add sound effects on top of it.

Inputs

USB mouse - libusb_open_device_with_vid_pid

```
dev_handle = libusb_open_device_with_vid_pid(ctx, 16700, 12314); //open m
rr = libusb_interrupt_transfer(dev_handle, 0x81, datain, 0x0004, &size, 0);
```

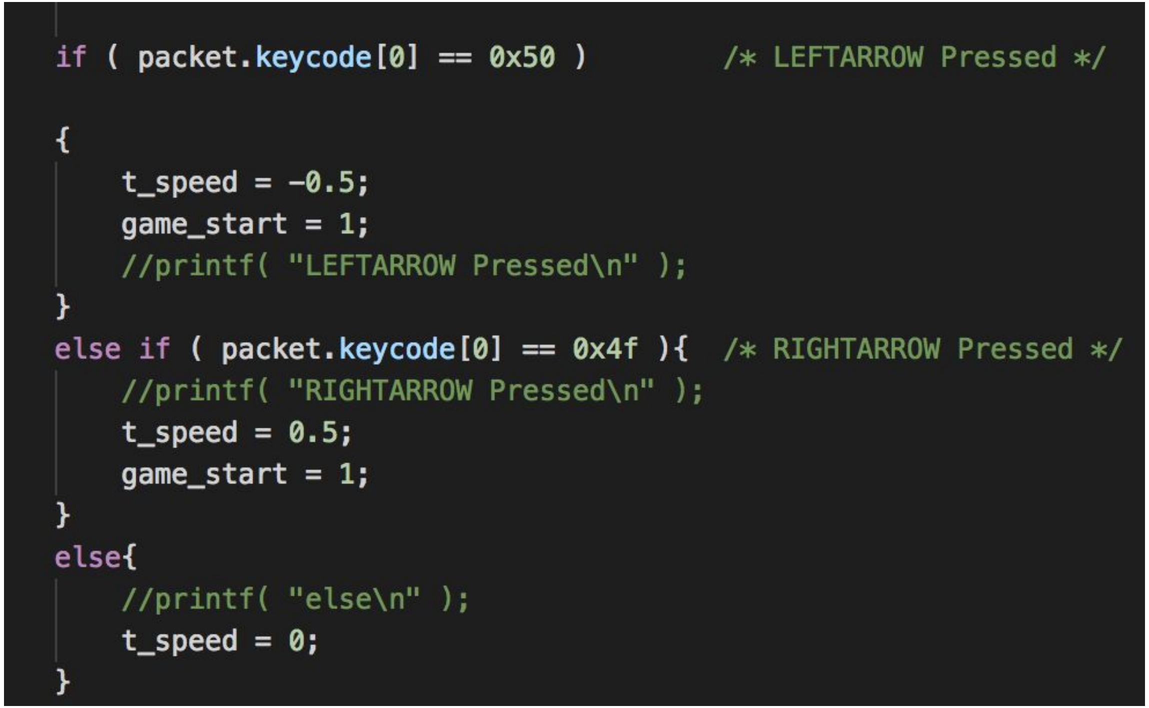
The mouse will return four bytes of data.
In this project only second and the last were used.



```
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 02 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 02 ff 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 00 00
libusb_interrupt_transfer :  0
size:  4
moving right
data: 00 01 ff 00
libusb_interrupt_transfer :  0
size:  4
scrolling up
data: 00 00 00 01
libusb_interrupt_transfer :  0
size:  4
scrolling down
data: 00 00 00 ff
```

Inputs

USB keyboard left and right



```
if ( packet.keycode[0] == 0x50 )        /* LEFTARROW Pressed */

{
    t_speed = -0.5;
    game_start = 1;
    //printf( "LEFTARROW Pressed\n" );
}
else if ( packet.keycode[0] == 0x4f ){  /* RIGHTARROW Pressed */
    //printf( "RIGHTARROW Pressed\n" );
    t_speed = 0.5;
    game_start = 1;
}
else{
    //printf( "else\n" );
    t_speed = 0;
}
```

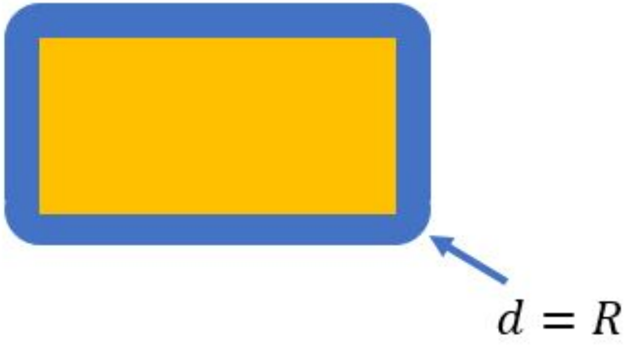
Core Parameters

- Ball location (x,y) (2*10 bits)
- Paddle location (10 bits)
- Brick status {brick_exists, brick_gone} (1*6*10 bits)
- Score (3*4 bits)
- Lives (2 bits)
- Game status {normal, won, lost} (2 bits)
- Audio control {normal, hit_wall, hit_brick} (2 bits)

General Game Logic

- 1. Programmable brick layout
- 2. Several Levels(difficulty) of games, 2 levels currently for faster demonstration
- 3. Press “ENTER” to start
- 4. Random initial direction, fixed absolute value of speed
- 5. 3 lives in total, shown on the top right corner
- 6. Score system: 2pts for green bricks, 1pt for blue brick
- 7. Reset after 3 lives are gone / player has passed all levels

Bouncing Models - hit



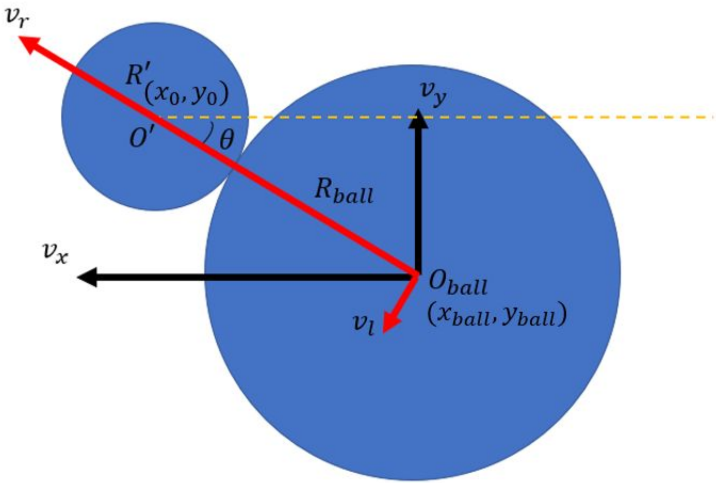
Hit on brick

Ball center falling into the blue region means the ball is hitting on the brick.

Hit on paddle

Similarly to determine whether the ball is hitting on the brick or not.

Bouncing Models - bounce



Hit on brick

- 1. Top, bottom $v_x' = v_x\,, v_y' = -v_y$
- 2. Right, left $v_x' = -v_x\,, v_y' = v_y$
- 3. Hit on corners, consider the corner as a circle with r=0. The radial speed vr = -vr, lateral speed vl = vl.
- Do some maths, it gives

$$\begin{cases} v_x' = -\cos(2\theta)\cdot v_x - \sin(2\theta)\cdot v_y \\ v_y' = -\sin(2\theta)\cdot v_x + \cos(2\theta)\cdot v_y \end{cases}$$

$$\theta = \tan^{-1}\frac{y_0 - y_{ball}}{x_0 - x_{ball}}$$

Thank you!