EEET2490 – Embedded System: OS and Interfacing, Semester 2023-2

Assessment 3 – Group Work Report

SCREEN DISPLAY & APPLICATION DEVELOPMENT FOR BARE METAL OS

Lecturers: Mr Linh Tran – linh.tranduc@rmit.edu.vn,

Mr. Phuc Nguyen - phuc.nguyenhoangthien@rmit.edu.vn

Team Number: Group 13

Team Members:

1. Phan Nhật Minh - s3904422
2. Trần Mai Nhung - s3879954
3. Trần Nguyễn Hà Khanh - s3877707
4. Nguyễn Ngọc Minh - s3907086

Date: 27/09/2023

TABLE OF CONTENTS

I. INTRODUCTION 1

II. SCREEN DISPLAY 1

1. BACKGROUND 1

2. IMPLEMENTATION 1

III. APPLICATION DEVELOPMENT 8

1. Game Overview 8

2. Game Component 8

3. Game Flow 11

4. Game Instruction 11

5. Game Desgin 14

a) Keyboard Input 18

b) Delay using hardware counter 18

c) Backup And Restore Background Image 18

d) Print word and sentences 20

e) Navigate menu arrow 20

f) Clear screen with drawRectARGB32 21

g) Background display 22

h) Main menu display, help menu display, and difficulty selection display 22

i) Set background state and Set bird state 23

j) Game state handler 24

k) End game animation with generate color code. 28

l) Game starting and running Logic 29

6. Implementation Results 40

IV. CONCLUSION 49

V. REFERENCES (USE IEEE STYLES) 49

I. INTRODUCTION

This document dives into our project where we developed an embedded operating system (OS) with capability to display static data and process interactive contents. In details, the first objective of this project is to design an OS that effortlessly displays images, plays videos, and presents text with custom font on a screen. By using the listed foundational features and applying our study in peripheral driver development, we took an extra step to accomplish our second objective, which is develop an embedded game that can run seamlessly on our OS.

The report is structured to thoroughly present our project, its features and implementation explanation, as well as the final outcome. Essential background information will be provided in advanced, and the next sections will illustrate our implementation approaches for each feature and development. Demonstration result will also be presented and discussed in this report.

II. SCREEN DISPLAY

1. BACKGROUND

❖ Display Image

It is possible to display an image on framebuffer screen using Raspberry Pi by drawing each pixel of the image to the framebuffer. Firstly, the image needs to be converted into an array of ARGB32 data, which is equivalent to BGRA8888 byte order on little-endian system (Raspberry Pi) [1]. In the array, each array element represents the color of the image at a pixel point. Therefore, the program can loop through each pixel location (x,y) and fill that pixel color with the corresponding element in the byte array. Finally, the program goes to the next pixel and increments the index in the array by 1 to draw it.

❖ Display Video

A video is just a collection of all of its frame images. In order to display a video, we can extract all of its frames, and then display each image one by one.

❖ Display Font

Displaying 40x40 pixel letters and characters with autolova font from the model [2] are required to do conversions from images to ARGB32 data. Then, it is necessary to scan through the data to display the correct font and color for each letter and character.

2. IMPLEMENTATION

a) Image and Video

❖ Image

The OS can display an image on screen for its users. When the user execute ***displayImage*** command in the CLI, they can press ‘w’ and ‘s’ keys to scroll up and down the image. Additionally, there are three different images available, which users can press ‘a’ and ‘d’ to switch backwards and forwards to see them.

To display an image, we need the array of its ARGB32 data. We use image2cpp [3] to convert the image into the type we need (choose RGB888 option).

```
You, 2 weeks ago | 1 author (You)
// 'cloud', 1080x675px        You, 2 weeks ago • 3 background to demo slideshow …
const unsigned long background_cloud [] = {
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000f9fb, 0x0000f9fb, 0x0000f9fb, 0x0000f9fb, 0x0000f9fb, 0x0000f9fb, 0x0000f9fb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000f9fb, 0x0000f8fb, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f8fb, 0x0000f9fb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000fafb, 0x0000f9fb, 0x0000f7fc, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f6fc, 0x0000f9fb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0000fafb, 0x0
  0x0000f7fc, 0x0000f7fc, 0x0000f6fd, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fd, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0000f7fc, 0x0
  0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0000f5fd, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
  0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0000f5fe, 0x0
```

Figure 1: Example of byte array of an image

drawImage() function draws an image at a location on the screen with the given byte array of the image, the width and height of the image, and the starting point coordination (x, y).

```
void drawImage(const unsigned long* bitmap, int width, int height, int x, int y) {
    int index = 0;
    for (int h = y; h < y + height; h++) {
        for (int w = x; w < x + width; w++) {
            drawPixelARGB32(w, h, bitmap[index]);
            index++;
        }
    }
}
```

Figure 2: Function to draw image

Starting from index 0 of the byte array, we draw the first pixel at the location (x, y). Then, we increase the index by 1 and go to the next pixel (next column, same row) and draw it. We keep repeating until all pixels (all rows and columns) of the image are filled with color.

Finally, to display the image on screen for the ***display_image()*** command, the team creates a function display_image() to draw the image with our designated size. In order to demonstrate the scroll up and scroll down feature, we set the physical screen to be smaller than the virtual screen, and draw the image with full width and height of the virtual screen from the top-left corner (x = 0, y = 0).

```
void display_image() {
  drawImage(background_allArray[current_bg], virScreenWidth, virScreenHeight, 0, 0);
}
```

Figure 3: Function to display image on screen for CLI command displayImage

While displaying the image, the program detects keypress event from the user.

```c
while (1) {        You, 2 days ago • combine with nhung …
  char c = uart_getc();
  if (c == '\n') break;
  else if (c == 'a') { // slide to previous image
    if (current_bg == 0) current_bg = background_LEN-1;
    else current_bg--;
    display_image();
  }
  else if (c == 'd') { // slide to next image
      if (current_bg == background_LEN-1) current_bg = 0;
      else current_bg++;
      display_image();
  }
  else if (c == 'w') { // scroll up
    scroll_up_image();
  }
  else if (c == 's') { // scroll down
    scroll_down_image();
  }
}
```

Figure 4: Logic of keypress-detecting when display image

If 'w' is pressed, it will scroll up the image, and if 's' is pressed, it will scroll down the image. To 'scroll up' and 'scroll down', the program indeed adjusts the vertical virtual offset (named *y_offset*) of the framebuffer. The program calls mailbox with the tag SET VIRTUAL OFFSET to change *y_offset*. The initial *y-offset* is 0, which means the physical screen is placed on the top edge of the virtual screen. The *y_offset* is increased by 1 to scroll up, and is decreased by 1 to scroll down. The program only allows to increase the *y_offset* if the physical screen has not reached the bottom edge of the virtual screen, which is to avoid physical screen from overflowing vertical screen. Similarly, the program only allows to decrease the *y_offset* if the physical screen has not reached the top edge of the virtual screen (*y_offset > 0*).

```c
void scroll_up_image() {
  if (y_offset > 0)  y_offset--;
  unsigned int *res_data = 0;
  mbox_buffer_setup(ADDR(mBuf), MBOX_TAG_SETVIRTOFF, &res_data, 8, 8, 0, y_offset);
  mbox_call(ADDR(mBuf), MBOX_CH_PROP);
}

void scroll_down_image() {
  if (y_offset + screenHeight < virScreenHeight) y_offset++;
  unsigned int *res_data = 0;
  mbox_buffer_setup(ADDR(mBuf), MBOX_TAG_SETVIRTOFF, &res_data, 8, 8, 0, y_offset);
  mbox_call(ADDR(mBuf), MBOX_CH_PROP);
}
```

Figure 5: Functions to scroll up and scroll down image

The program also detects keys 'a' and 'd' to switch backwards and forwards three images.

To implement this feature, we firstly convert each image into byte data, and then put them together into a whole array called *background_allArray[]* whose size is *background_LEN*. We also have a variable named *current_bg* to determine the image to display. *current_bg* represents the index of the image in the *background_allArray[]*, and it is initialized with a default value to ensure that the program has an image to display by default.

```
// background
const int background_LEN = 3;
const unsigned long* background_allArray[3] = {
  background_sky,
  background_sun,
  background_cloud
};
```

```
int current_bg = DEFAULT_BACKGROUND;
```

Figure 6: Variables to make switch between images feature

To display an image at index *X*, we set *current_bg = X* so that *background_allArray[current_bg]* in funtion *display_image()* (Figure XXX) will point to the address of the byte array of the image indexed X.

To switch between the images, when the user presses 'a', *current_bg* is decreased by 1 to go to the previous index; otherwise, when the user presses 'd', *current_bg* is increased by 1 to go to the next index. We also have conditional statements to ensure that the index is not out of array's size. After *current_bg* is set, the program calls *display_image()* to display new image on the screen.

```
else if (c == 'a') { // slide to previous image
  if (current_bg == 0) current_bg = background_LEN-1;
  else current_bg--;
  display_image();
}
else if (c == 'd') { // slide to next image
    if (current_bg == background_LEN-1) current_bg = 0;
    else current_bg++;
    display_image();
}
```

Figure 7: Logic to switch between images

❖ Video

To display a video, we firstly extract the frame images from the video using "ffpmeg" [4] with 24 frames per second. The reason why I choose this frame rate is because there is limitation when compiling the program in Windows. Therefore, after testing with various frame rate, we choose the frame rate to be 24 with 3-second video. Then, we store all the frames into an array, and loop through the array to draw each frame.

```
// first video
const int first_video_array_LEN = 75;
const unsigned long* first_video_array[75] = {
    first_video_flappy_frames_frame_flappy_0001,
    first_video_flappy_frames_frame_flappy_0002,
    first_video_flappy_frames_frame_flappy_0003,
    first_video_flappy_frames_frame_flappy_0004,
    first_video_flappy_frames_frame_flappy_0005,
    first_video_flappy_frames_frame_flappy_0006,
    first_video_flappy_frames_frame_flappy_0007,
    first_video_flappy_frames_frame_flappy_0008,
    first_video_flappy_frames_frame_flappy_0009,
    first_video_flappy_frames_frame_flappy_0010,
    first_video_flappy_frames_frame_flappy_0011,
    first_video_flappy_frames_frame_flappy_0012,
    first_video_flappy_frames_frame_flappy_0013,
    first_video_flappy_frames_frame_flappy_0014,
    first_video_flappy_frames_frame_flappy_0015,
    first_video_flappy_frames_frame_flappy_0016,
    first_video_flappy_frames_frame_flappy_0017,
    first_video_flappy_frames_frame_flappy_0018,
    first_video_flappy_frames_frame_flappy_0019,
```

Figure 8: Frames array of a video

We create function ***drawVideo()*** to draw a video to the screen. The function takes the array containing all frames of the video and the number of the total frames then start drawing each frame image until the last frame. There is a delay duartion between each frame to make an appropriate speed for the video. Additionally, the function also provides options to repeat the video or just play it once. If the input parameter *is_infinite* is 1, the video will be replayed whenver it reaches the end; otherwise, if *is_infinite* is 0, the video will only be played once and then stops.

```
void drawVideo(const unsigned long* videoArray[], int num_frames, int img_width, int img_height, int is_infinite) {
    // is_infinite: if users want to loop the video - 1, if users dont want - 0
    // define index for looping frames
    int i = 0;
    while (1) {
        // if reaching final frame
        if (i == num_frames) {
            if (is_infinite == 1) {
                i = 0;
            }
            else {
                return;
            }
        }
        // draw image
        drawImage(videoArray[i], img_width, img_height, 0, 0);  // Draw the image at the top-left corner
        // delay(FRAME_DURATION_VIDEO);  // Delay for the next frame
        wait_msec(1000 / FRAME_DURATION_VIDEO);
        i++;
    }
}
```

Figure 9: Function to draw a video

When the user executes ***displayVideo*** command from the CLI, the program calls function ***display_video()*** to display the video to the framebuffer screen.

```
void display_video() {
  drawVideo(first_video_array, first_video_array_LEN, 480, 636, 0);
  wait_msec(100);
}
```

Figure 10: Function to display video on screen for CLI command displayVideo

b) Font

To display the full name of all 4 team members to the screen, we snip each individual alphabet letter and process it via “image2cpp” [2] to extract the 32-bit hexacode for each pixel of the letter that will be displayed on the screen. During the conversion, we decided that each letter and character would consist 40x40 pixels, and each pixel would contain 3 bytes. The ARGB32 data of every letters and characters are stored in an 2-dimensional array called “fontData”.

```
// 40x40 px
unsigned int fontData[128][2500] = {
        ['A'] =
                {
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e5
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00c8
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e5e5e5, 0x0039
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00cdcdcd, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e3e3e3, 0x002f2f2f, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e3e3e3, 0x00d8d8d8, 0x00d2d2d2, 0x00d4d4d4, 0x00dadada, 0x006f6f6f, 0x00000000, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00cdcdcd, 0x00919191, 0x00363636, 0x00050505, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e5e5e5, 0x00c0c0c0, 0x00424242, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e5e5e5, 0x005d5d5d, 0x00020202, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00dfdfdf, 0x00373737, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00020202, 0x002e2e2e, 0x00929292, 0x00c0c0c0, 0x00c9c9c9, 0x00262626, 0x00000000, 0x00000000, 0x00000000, 0x0001
                        0x00e6e6e6, 0x00e6e6e6, 0x00e4e4e4, 0x00383838, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00030303, 0x009a9a9a, 0x00e4e4e4, 0x00e6e6e6, 0x00e6e6e6, 0x00949494, 0x00000000, 0x00000000, 0x00000000, 0x0034
                        0x00e6e6e6, 0x00e6e6e6, 0x00878787, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x005c5c5c, 0x00e6e6e6, 0x00e6e6e6, 0x00e5e5e5, 0x00a4a4a4, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00010101, 0x00c1
                        0x00e6e6e6, 0x00e2e2e2, 0x000f0f0f, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x001c1c1c, 0x008c8c8c, 0x008e8e8e, 0x00464646, 0x00030303, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00191919, 0x00e6
                        0x00e6e6e6, 0x00d1d1d1, 0x00010101, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00010101, 0x00afafaf, 0x00e6
                        0x00e6e6e6, 0x009a9a9a, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00585858, 0x00e5e5e5, 0x00e6
                        0x00e6e6e6, 0x00afafaf, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00303030, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00dcdcdc, 0x00050505, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00404040, 0x00e2e2e2, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00707070, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00010101, 0x007d7d7d, 0x00e3e3e3, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e4e4e4, 0x07747474, 0x000a0a0a, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x000d0d0d, 0x004b4b4b, 0x00c4c4c4, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00d4d4d4, 0x007f7f7f, 0x00454545, 0x003c3c3c, 0x005b5b5b, 0x00a1a1a1, 0x00d9d9d9, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
                        0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6e6e6, 0x00e6
        ['B'] =
                {
```

Figure 11: The ARGB32 data of letter “A” is stored in a 2D array called “fontData”

In “framebf.c”, a function called “drawLetter” is built to display letters and characters individually. The function takes 4 arguments: the character “ch” to be displayed, the starting “x” and “y” coordinates of the top-left corner of the character will be positioned, and a “colorCode” that sets the color of the character. A nested for loop is used to scan through the 40x40 pixel area. If the corresponding pixel in the fontData is set to 0x00000000, it will be colored using the provided color code from the “colorCode” parameter. The increasing value of the “indexCount” variable helps the function keeps track of which pixel in the “fontData” array it’s currently examining.

```
void drawLetter(char ch, int x, int y, unsigned int colorCode) {
    int indexCount = 0;
    //Nest loop run through 40x40 pixel area
    for (int i = y; i < y + 40;i++) {
        for (int j = x; j < x + 40; j++) {
            if (fontData[(unsigned int)ch][indexCount] == 0x00000000) {
                drawPixelARGB32(j, i, colorCode);
            }
            indexCount++;
        }
    }
}
```

Figure 12: The “drawLetter” function is used to display each letter

Then, we repeatedly call this function to display the name of all team members as the requirement.

```
//Group name
drawLetter('P', 20, 20, 0x009400D3);
drawLetter('h', 60, 20, 0x004B0082);
drawLetter('a', 100, 20, 0x000000FF);
drawLetter('n', 140, 20, 0x0000FF00);
drawLetter('N', 200, 20, 0x00FFFF00);
drawLetter('h', 240, 20, 0x00FF7F00);
drawLetter('a', 280, 20, 0x00FF0000);
drawLetter('t', 320, 20, 0x009400D3);
drawLetter('M', 380, 20, 0x004B0082);
drawLetter('i', 420, 20, 0x000000FF);
drawLetter('n', 460, 20, 0x0000FF00);
drawLetter('h', 500, 20, 0x00FFFF00);

drawLetter('T', 20, 80, 0x00FF7F00);
drawLetter('r', 60, 80, 0x00FF0000);
drawLetter('a', 100, 80, 0x009400D3);
drawLetter('n', 140, 80, 0x004B0082);
drawLetter('M', 200, 80, 0x000000FF);
drawLetter('a', 240, 80, 0x0000FF00);
drawLetter('i', 280, 80, 0x00FFFF00);
drawLetter('N', 340, 80, 0x00FF7F00);
drawLetter('h', 380, 80, 0x00FF0000);
drawLetter('u', 420, 80, 0x009400D3);
drawLetter('n', 460, 80, 0x004B0082);
drawLetter('g', 500, 80, 0x000000FF);

drawLetter('N', 20, 140, 0x0000FF00);
drawLetter('g', 60, 140, 0x00FFFF00);
drawLetter('u', 100, 140, 0x00FF7F00);
drawLetter('y', 140, 140, 0x00FF0000);
drawLetter('e', 180, 140, 0x009400D3);
drawLetter('n', 220, 140, 0x004B0082);
drawLetter('N', 280, 140, 0x000000FF);
drawLetter('g', 320, 140, 0x0000FF00);
drawLetter('o', 360, 140, 0x00FFFF00);
drawLetter('c', 400, 140, 0x00FF7F00);
drawLetter('M', 460, 140, 0x00FF0000);
drawLetter('i', 500, 140, 0x009400D3);
drawLetter('n', 540, 140, 0x004B0082);
drawLetter('h', 580, 140, 0x000000FF);

drawLetter('T', 20, 200, 0x0000FF00);
drawLetter('r', 60, 200, 0x00FFFF00);
drawLetter('a', 100, 200, 0x00FF7F00);
drawLetter('n', 140, 200, 0x00FF0000);
```

Figure 13: Repeatedly call the "drawLetter" function to display the full name

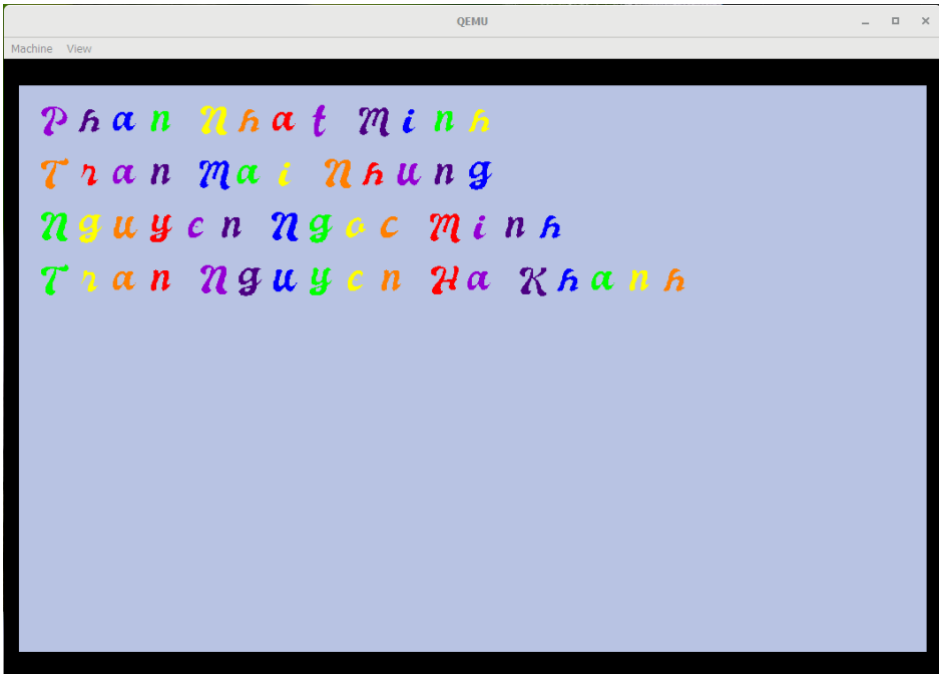

Figure 14: All 4 names of the team members are displayed with different color for each letter

III. APPLICATION DEVELOPMENT

1. Game Overview

Our team has developed a game called "Bird Bay Lag", which is based upon a worldwide viral arcade game called "Flappy Bird" by Nguyễn Hà Đông [5]. We decided that it is essential to improve and implement more features into our game. "Bird Bay Lag" 's rules are simple: press key SPACE to fly the bird, and the game ends when the bird hits an obstacle. The game has 3 difficulty levels, which are less, normal, and extreme. We add another type of obstacle to make the game more impressive to the gamers with all of ages. This type of obstacle has only 1 direction – North West direction – with a diversity of appearances. Also, there is a noticeable feature where the player can select different skins for the bird and background for a friendlier gameplay experience.

2. Game Component

There are 3 categories of component are assembled to build the game: background, obstacles, and player.

The background images is always set to be the same size as the physical screen:

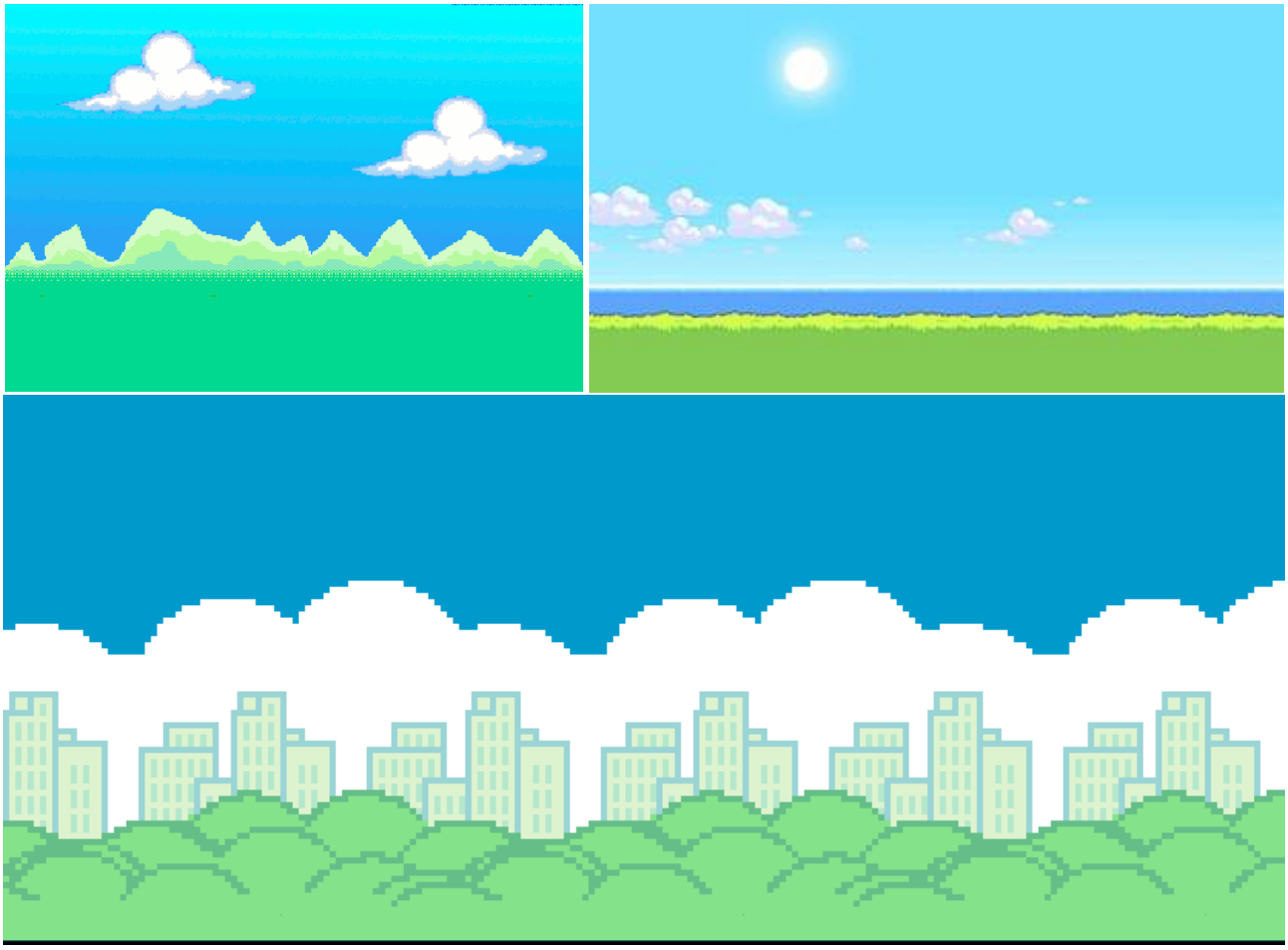

Figure 15: 3 different backgrounds corresponding to 3 difficulty levels

There are various of obstacles for the player to overcome. Noticably, the height dimension of the pipes will be randomly generated to create difficulties for the player.

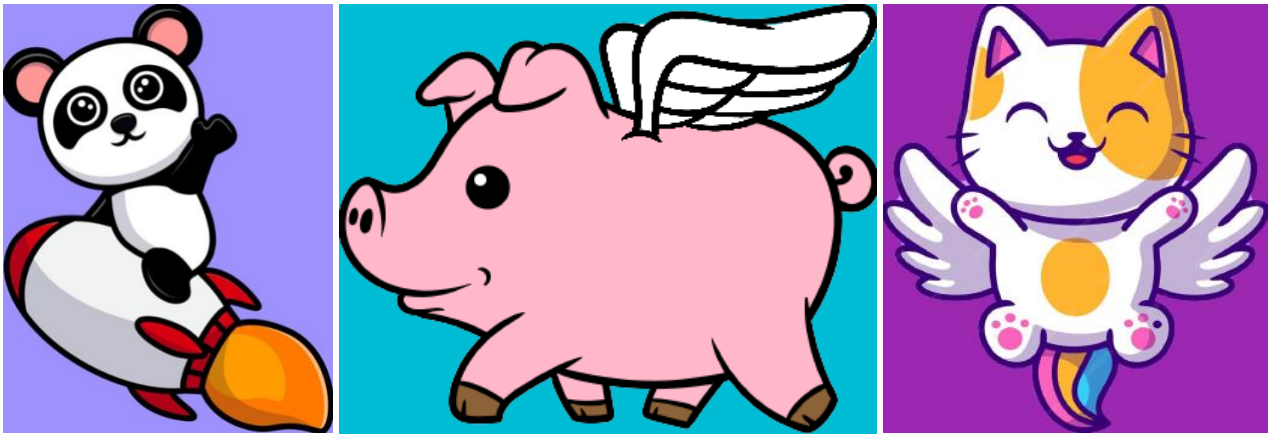

Figure 16: 8 obstacles with various size in width and height

The bird has 1 default skin and 5 alternative skins:

Figure 17: Default skin of the bird

Figure 18: 5 alternative skins for the bird

3. Game Flow

The game has 5 main states

- Main menu

At this state, the user is required to choose an option from the menu, which includes Start, Help, Exit. If Help is chosen, the game will show a screen displaying the game instructions for the user. If Exit is chosen, the program will exit the game. Otherwise, if Start is chosen, the user will be transferred to next state, which is set background.

- Set background

At this state, the user can choose their preferred background among different backgrounds. The chosen background will be displayed when they play game.

- Set bird

After setting background, the user can select the bird they prefer among different available bird skins.

- Set difficulty level

One last step before playing the game is to choose the difficulty level.

- Play game

When all above states are done, the program moves to playGame state where the user can play the game. When the game is over (the user lost or finished all the levels), the game will be turned back to Main menu state.

4. Game Instruction

Press 'w' or 's' to navigate through the main menu.
Press Enter key to select.

Figure 19: Welcome screen (main menu)

Press ‘a’ or ‘d’ to display different backgrounds.
Press Enter key to select background image.

Figure 20: Set Background screen

Press ‘a’ or ‘d’ to display different bird costume.
Press Enter key to select bird skin.

Figure 21: Set Bird screen

Press ‘w’ or ‘s’ to navigate through the difficult selection.
Press Enter key to select difficulty.

Figure 22: Set Difficulty Level screen

Tap Space to make the bird fly.

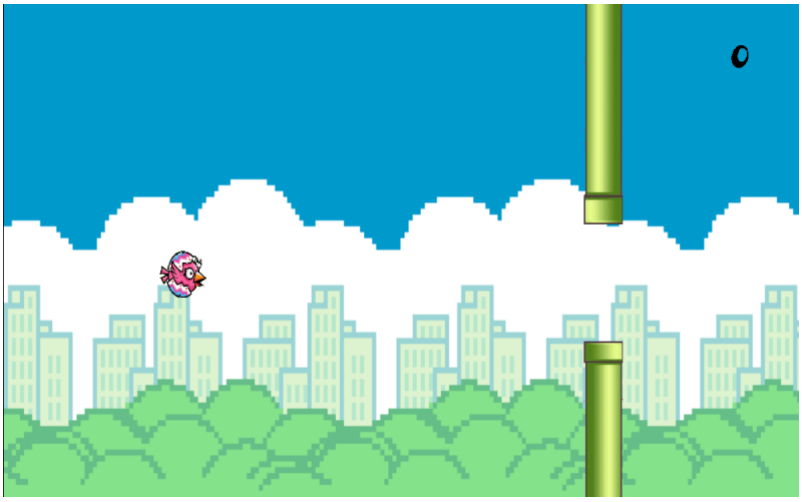

Figure 23: Play Game screen (1)

Get points for every obstacle you dodge.

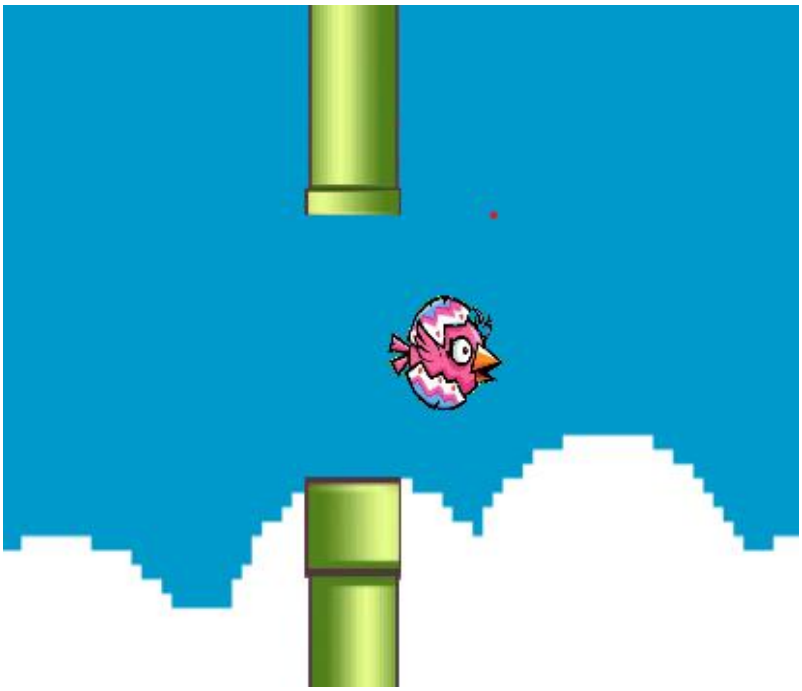

Figure 24: Play Game screen (2)

Don’t crash into an obstacle or you will lose.

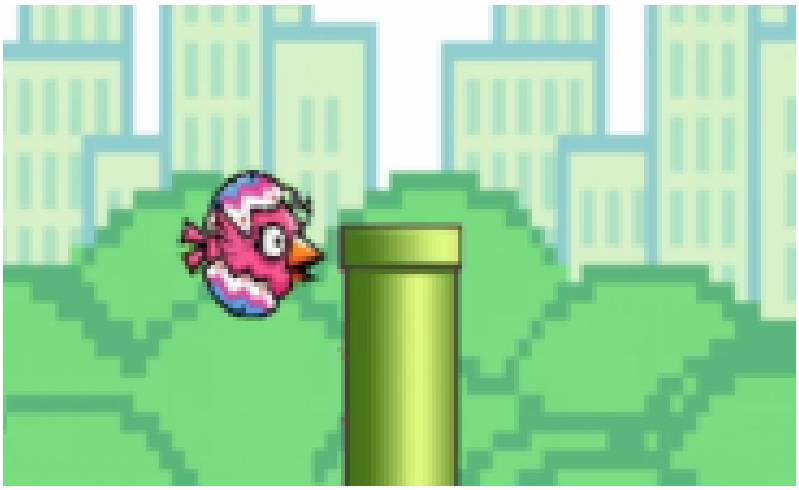

Figure 25: Play Game screen (3)

Press any key to get back to the main menu.
Play again and try to get higher score.

Figure 26: Game Over screen

5. Game Desgin

This section divides into 2 main parts: menu and setting part, and game playing part.

The figure below shows the main game flow chart:

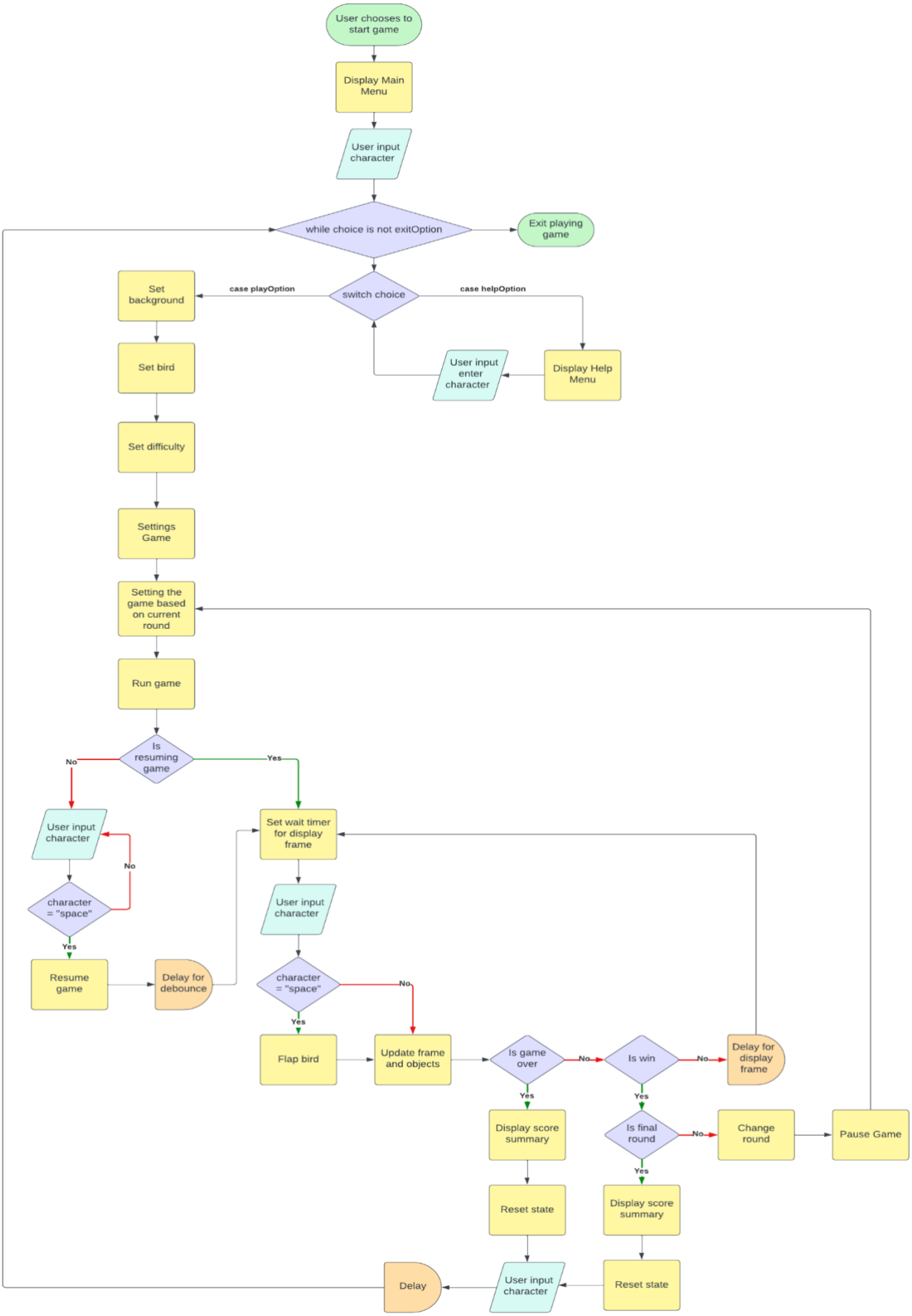
User chooses to start game
Display Main Menu
User input character
while choice is not exitOption
Exit playing game
switch choice
case playOption
case helpOption
Set background
Set bird
Set difficulty
Settings Game
Setting the game based on current round
Run game
Display Help Menu
User input enter character
Is resuming game
No
Yes
User input character
No
character = "space"
Yes
Resume game
Delay for debounce
Set wait timer for display frame
User input character
character = "space"
No
Yes
Flap bird
Update frame and objects
Is game over
No
Yes
Is win
No
Yes
Delay for display frame
Display score summary
Is final round
No
Yes
Change round
Pause Game
Reset state
Display score summary
Reset state
User input character
Delay

Figure 27. Flow chart of whole game system

This figure shows how main menu displays before the user selects playGame option. Firstly, main menu prompts the user to input a character to perform navigating actions. If the user chooses Exit option, the program backs the user to the terminal or the console. If the user chooses help option, there will be a instruction screen and the user can back to main menu by pressing enter. Finally, if the user chooses game option, then, the user must chooses the background, bird skin, and difficulty level before starting game. The details of starting game is discussed in below chart.

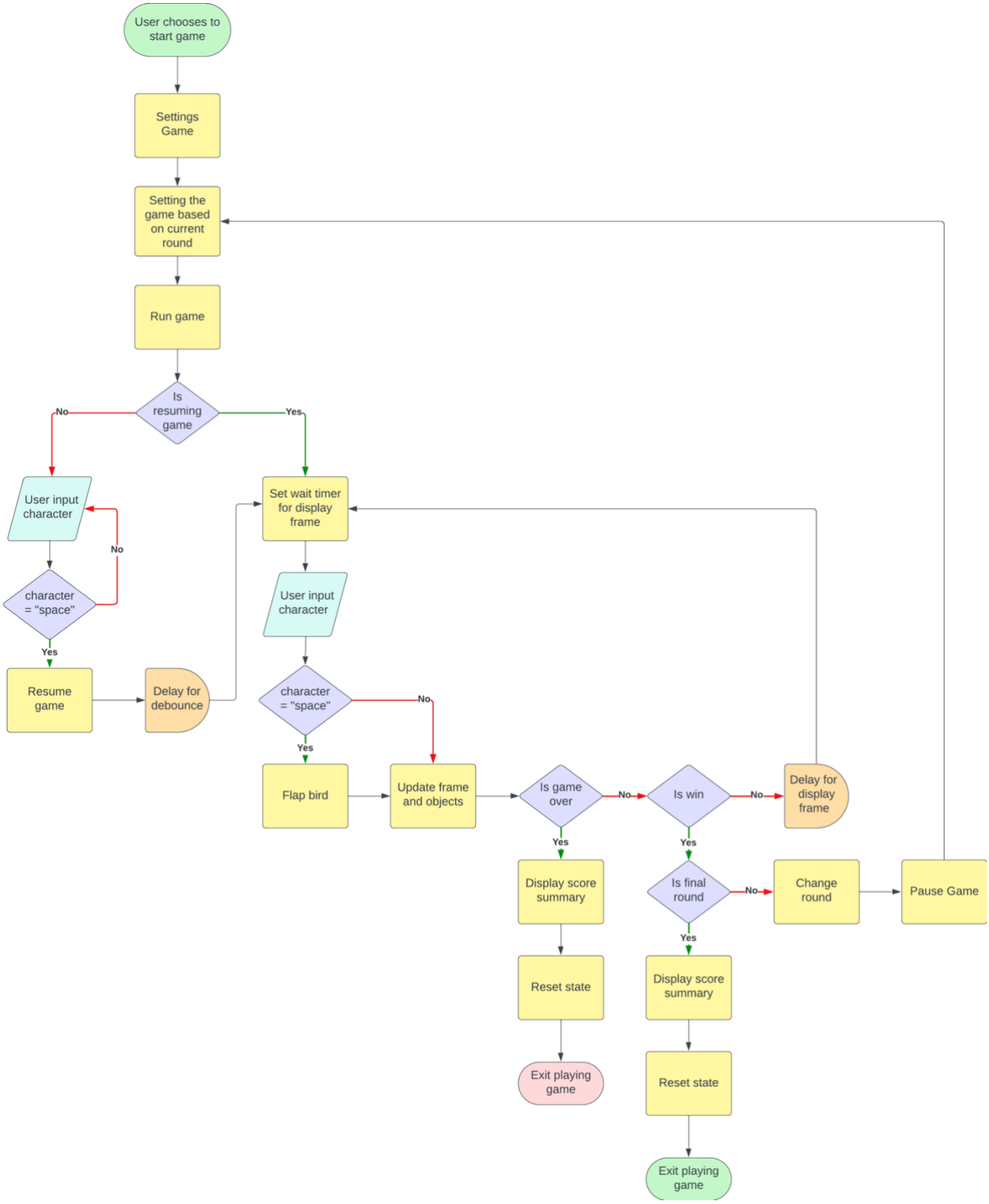

Figure 28 Game's flow chart

The figure above shows the game running flow chart. After user finishes choosing difficulty, bird, and background, the program will implement general settings for the game, including:

- Clear old screen and background
- Settings the game based on difficulty level

- Reset game scores
- Draw background and backup the background for restoring later when performing animation in game
- Reset current gaming round to be the first round
- Reset other game characteristics to default.

After finishing running setting, the program jumps to setting the game based on current round. In each round, the program converts scores from integer to string to display onto the game screen, defining the obstacles types and their moving directions based on round level. Next, the program generates a list of obstacles, including width, height, x, and y positions. If the obstacle is a pipe type, the program generates the y positions of both top and bottom pipes. Then, the program draws the pipes whose positions are inside the screen width. The program also initializes the bird position to be in the middle of the screen, its initial velocity to be 0, and drawing the bird onto the game screen. Afterwards, the game starts running.

To prevent immediately starting the game without making the user noticable and lose, the game is in the `pause` state until the user inputs `space` character. There is also debouncing delay to prevent bouncing. The game officially starts. The program will update the bird, obstacle movements, and current scores after passing each obstacle continuously until gaming over or gaming wind. If the bird reaches borders of the screen, or collides with the obstacles, the game is over. If the bird passes all required obstacles in each round, the game is won. If current round is the last round, the program navigates the user to summary score screen and ends game. Otherwise, the program jumps to next round and back to the setting game based on current round step. Finally, the program delays for **8.5 fps** (frames per second rate) to display game animation. The reason why the team chooses this rate instead of any common rates [6] because is this C program is synchronous, which runs the tasks one by one. Therefore, the program takes time to clear old objects and redraw objects with new position. Especially, this game has background image, instead of one color background.

The details of this playing game flow will be in `Game starting and running Logic`.

a) Keyboard Input

We use UART 1 to take input

- ***uart_getc()*** checks if a key is pressed. It waits until the data is ready for read and convert carriage '\r' to new line.
- ***getUart()*** checks if a key is pressed and return 0 when there's no key press. Using this function, the program does not need to wait for a key pressed, it can run other things if the user presses nothing.

b) Delay using hardware counter

There are 2 functions used in this program:

- ***wait_msec():*** this function is simpler than another one. This function simply sets the clock counting rate by calculating frequency and counter value, and blocks the program to loop the counter value until it is larger than the calculated counting rate
- ***set_wait_timer():*** this function is when you want to have accurate delay time, and when you need to implement some tasks between setting and running the timer. Therefore, to use this function, firstly, call function with mode 1 to immediately exit the function after calculation. Then, implement the tasks. Finally, call mode 2 to run the timer. This is useful when the program wants to delay time before displaying new frames to have animation or videos.

c) Backup And Restore Background Image

We design two functions ***backupRegion()*** and ***clearImageOverlay()*** to receive an area pixel for backup and restore when needed.

- ***backupRegion()*** function takes *the x* and *y* values from the top left corner with *the width* and *height* of the area pixels we want to back up. It goes through each pixel in this area using a nested loop and stores its value in a buffer array named *backupBuffer*.
    - The start values inside the nested loop are the x and y positions counting from set offsets when setting the frame buffer. The end values inside the nested loop are the width and height measuring starting from positions x and y.
    - Because the team project cannot use a standard library, the team let the backupBuffer have a maximum size $MAX_WIDTH * MAX_HEIGHT$. This array represents a 2D array with physical screen width and physical screen height size after flatting it into a 1D array. Each element in backupBuffer represents one pixel storing one color code in the window.
    - **How it works**. Firstly, the function loops through the mentioned nest loop. Inside the loop, the program defines the position of pixel where the program needs to backup. Next, the program gets the offset of the pixel in original image in the frame buffer, and then gets to pixel value from them. Afterwards, the program stores the pixel value back to the backupBuffer.

```
#define MAX_WIDTH 1080   // maximum expected width
#define MAX_HEIGHT 675  // maximum expected height
// Backup buffer for the screen
uint32_t backupBuffer[MAX_WIDTH * MAX_HEIGHT];
```

Figure 29: Define max width and max height of the game screen

```
void backupRegion(int x, int y, int width, int height) {
    // Init backup index
    int backupIndex = 0;
    // Loop through the region
    for (int h = y; h < y + height; h++) {
        for (int w = x; w < x + width; w++) {
            // Calculate the index of the pixel in the backup buffer
            backupIndex = h * MAX_WIDTH + w;
            // Calculate the offset of the pixel in the frame buffer
            int offs = (h * pitch) + (COLOR_DEPTH / 8 * w);
            // Get the pixel value from the frame buffer
            uint32_t pixel_value = *((unsigned int*)(fb + offs));
            // Store the pixel value in the backup buffer
            backupBuffer[backupIndex] = pixel_value;
        }
    }
}
```

Figure 30: backupRegion()

- ***clearImageOverlay()*** function restores the area pixel to its previous state using *backupBuffer*. It received the same parameter as backupRegion() function and traverse through the pixel area to draw the color from the *backupBuffer*.

```
void clearImageOverlay(int x, int y, int width, int height) {
    // loop through the region
    for (int h = y; h < y + height; h++) {
        for (int w = x; w < x + width; w++) {
            // Get the pixel value from the backup buffer
            unsigned int originalColor = backupBuffer[h * MAX_WIDTH + w];
            // Draw the pixel
            drawPixelARGB32(w, h, originalColor);      You, last week via F
        }
    }
}
```

Figure 31: clearImageOverlay()

d) Print word and sentences

We create drawWord and drawSentence functions to render string on screen using the ***drawLetter()*** function. Each character is 40x40 pixel.

- ***drawWord()*** takes a word, color code and coordinates (x,y) then draws each letter from the left to right start from coordinate (x,y) until the end of string.

```
void drawWord(const char* word, int x, int y, unsigned int colorCode) {
    int offset = 0;
    for (int i = 0; word[i] != '\0'; i++) {
        drawLetter(word[i], x + offset, y, colorCode);
        offset += 40; // assuming each letter is 40 pixels wide
    }
}
```

Figure 32: drawWord()

- ***drawSentence()*** takes the same parameter as drawWord() function. Unlike drawWord() function, it handle the character space ' ' by not drawing anything and move the offset by 40 pixels wide.

```
void drawSentence(const char* word, int x, int y, unsigned int colorCode) {
    int offset = 0;
    for (int i = 0; word[i] != '\0'; i++) {
        if (word[i] != ' ') drawLetter(word[i], x + offset, y, colorCode);
        offset += 40; // assuming each letter is 40 pixels wide
    }
}
```

Figure 33: drawSentence()

e) Navigate menu arrow

We create two functions ***displayArrow()*** and ***deleteArrow()*** for displaying and delete an arrow graphic. This arrow is used as menu navigation and difficulty selection. The arrow image pixels array (*const unsigned long arrow[]*) with white background is defined inside the arrow.h.

- ***displayArrow()*** displays the arrow on the screen at the specified coordinate x and y. The function loops through each pixel color in the arrow[] array and compare to display an arrow with arrow color code when the color is not white (0x00FFFFFF).

```
// 'arrow', 80x40px
void displayArrow(const unsigned long* arr, int x, int y) { //
    for (int i = 0; i < 40; i++) {
        for (int j = 0; j < 80; j++) {
            if (*arr != 0x00FFFFFF) {
                drawPixelARGB32(x + j, y + i, arrowColorCode);
            }
            arr++;
        }
    }
}
```

Figure 34: displayArrow()

- ***deleteArrow()*** calls the function clearImageOverlay() to restore the color pixel that has been writeover by the displayArrow() function.

```
//Delete the arrow in the menu screen
void deleteArrow(int x, int y) {
    clearImageOverlay(x, y, 80, 40);
}
```

Figure 35: deleteArrow()

f) Clear screen with drawRectARGB32

clear_screen() function fills entire screen with black pixel color. It sends the top-left coordinate (0, 0), the bottom-right coordinate, color attribute and a fill flag = 1 to the drawRectARGB32() function. If the fill flag is = 0, the drawRectARGB32() function draws the outline of the rectangle.

```
void clear_screen() {
    drawRectARGB32(0, 0, virScreenWidth, virScreenHeight, 0x00000000, 1);
}
```

Figure 36: clear_screen()

```
void drawRectARGB32(int x1, int y1, int x2, int y2, unsigned int attr, int fill) {
    for (int y = y1; y <= y2; y++)
        for (int x = x1; x <= x2; x++) {
            if ((x == x1 || x == x2) || (y == y1 || y == y2))
                drawPixelARGB32(x, y, attr);
            else if (fill)
                drawPixelARGB32(x, y, attr);
        }
}
```

Figure 37: drawRectARGB32()

g) Background display

backgroundDisplay() function calls the drawImage() function to draw the background image with full-screen size, starting from the top-left corner.

```
//Function display background image
void backgroundDisplay() {
    drawImage(background_allArray[current_bg], virScreenWidth, virScreenHeight, 0, 0);
}
```

Figure 38: backgroundDisplay()

h) Main menu display, help menu display, and difficulty selection display

mainMenuDisplay(), ***helpMenuDisplay()*** and ***difficultSelectDisplay()*** draw the background image using backgroundDisplay() method. They call the backupRegion() to prepare the pixel for arrow navigate deletion and then draw instruction words using drawWord() and drawSentence() methods.

```
//Display main menu
void mainMenuDisplay() {
    backgroundDisplay();
    backupRegion(0, 0, screenWidth, screenHeight);
    drawWord("Welcome", 300, 150, startColorCode);
    drawWord("To", 620, 150, startColorCode);
    drawWord("Bird", 200, 250, startColorCode);
    drawWord("Bay", 460, 220, startColorCode);
    drawWord("LAG", 700, 280, startColorCode);

    drawWord("Start", 400, 350, startColorCode);
    drawWord("Help", 400, 400, startColorCode);
    drawWord("Exit", 400, 450, startColorCode);
}
```

Figure 39: mainMenuDisplay()

```
//Display help menu
void helpMenuDisplay() {
    backgroundDisplay();
    drawSentence("Instruction", 300, 100, helpColorCode);
    drawSentence("Press Space To Play", 140, 200, helpColorCode);
    drawSentence("Dodge Obstacles To", 140, 300, helpColorCode);
    drawSentence("Gain Point", 340, 350, helpColorCode);
    drawSentence("Back", 420, 430, helpColorCode);
}
```

Figure 40: helpMenuDisplay()

```
void difficultSelectDisplay() {
    backgroundDisplay();
    backupRegion(0, 0, screenWidth, screenHeight);
    drawSentence("LAG Selection", 240, 100, helpColorCode);
    drawSentence("Less", 380, 250, helpColorCode);
    drawSentence("Normal", 380, 350, helpColorCode);
    drawSentence("Extreme", 380, 450, helpColorCode);
}
```

Figure 41: difficultSelectDisplay()

i) Set background state and Set bird state

Similarly to the idea of switching between images that we have presented in the *Section 2a*, we initialize two variables called ***current_bg*** and ***current_bird*** to determine the index of the chosen background and the chosen bird to display. When the user presses key 'a'/'d', we will increase/decrease the index by 1 to go to next/previous image, and display the screen again. When the user presses ENTER, the game will move to the next state.

```
//Display instruction to select background
void setBackgroundStateDisplay() {
    backgroundDisplay();
    drawSentence("Instruction", 300, 100, helpColorCode);
    drawSentence("Press 'a' or 'd' to slide", 20, 200, helpColorCode);
    drawSentence("Enter to set background", 30, 300, helpColorCode);
}

//Display instruction to select bird costume
void setBirdStateDisplay() {
    backgroundDisplay();
    drawSentence("Instruction", 300, 100, helpColorCode);
    drawSentence("Press 'a' or 'd' to slide", 20, 200, helpColorCode);
    drawSentence("Enter to set bird", 160, 300, helpColorCode);
    // Draw bird
    draw_bird(bird, 120, 100);

}
```

Figure 42: setBackgroundStateDisplay() and setBirdDisplay()

```
case setBackground:
    c = getUart();
    if (c == '\n') {
        nextState = setBird;
    }
    else if (c == 'a') { // slide to previous image
        if (current_bg == 0) current_bg = background_LEN - 1;
        else current_bg--;
        setBackgroundStateDisplay();
    }
    else if (c == 'd') { // slide to next image
        if (current_bg == background_LEN - 1) current_bg = 0;
        else current_bg++;
        setBackgroundStateDisplay();
    }
    break;

case setBird:
    c = getUart();
    if (c == '\n') {
        nextState = setDifficult;
    }
    else if (c == 'a') { // slide to previous image
        if (current_bird == 0) current_bird = bird_allArray_LEN - 1;
        else current_bird--;
        setBirdStateDisplay();
    }
    else if (c == 'd') { // slide to next image
        if (current_bird == bird_allArray_LEN - 1) current_bird = 0;
        else current_bird++;
        setBirdStateDisplay();
    }
```

Figure 43: Keypress detection in setBackground state and setBird state

j) Game state handler

gameMenu() function implement a state-based menu for the game. We initializes 3 variables:

- arrowPos for tracking arrow position
- currState manage to display the current view.
- nextState managing transition between views.

```
int arrowPos = mainMenu;
int currState = mainMenu;
int nextState = mainMenu;
```

Figure 44: Game state handler – variables

The while(1) loop to ensure the view continue to rotation until the user choose exitOption.

The first ***switch(nextState)*** - *figure below* - displays the game view base on nextState. It then reposition arrowPos for display the corresponding arrow image next to the selection for user navigation. It sets the currState to the corresponding value that are displayed and reset the nextState value. After initialize and displaying views, it will call corresponding functions for setting up gameplay.

- mainMenu: display game menu.
- helpMenu: display help menu.
- setBackground: display option to select background images.
- setBird: display option to select bird appearance.

- setDifficult: display option to select game difficulty
- playGame: setting up gameplay environment, which will be discussed in detail in section `Game starting and running Logic`.

```
//Display corresponding views and reset nextState variable.
switch (nextState) {
case mainMenu:
    mainMenuDisplay();
    arrowPos = playOption;
    nextState = 0;
    currState = mainMenu;
    break;
case helpMenu:
    helpMenuDisplay();
    nextState = 0;
    currState = helpMenu;
    break;
case setBackground:
    setBackgroundStateDisplay();
    nextState = 0;
    currState = setBackground;
    break;
case setBird:
    set_bird_position((screenWidth / 2) - 130, 430);
    setBirdStateDisplay();
    nextState = 0;
    currState = setBird;
    break;
case setDifficult:
    difficultSelectDisplay();
    nextState = 0;
    currState = setDifficult;
    break;
case playGame:
    clear_screen();
    init_difficulty_setting();
    // init
    game_scores = 0;
    backgroundDisplay();
    backupRegion(0, 0, screenWidth, screenHeight);

    reset_round();
    reset_characteristics();
    nextState = 0;
    currState = playGame;
    break;
default:
    break;
}
```

Figure 45: Switch block to switch between game states

The second ***switch(currState)*** - *figure below* - waits for the user to input and handles it based on current state.

- mainMenu:
    - Navigate through options with ‘w’ key to move up and ‘s’ key to move down.
    - Press Enter key for selection.

- helpMenu:
    - Press Enter key to return to mainMenu.
- setBackground and setBird:
    - Navigate through background and bird's appearance options with 'a' key to move left and 'd' key to move right.
    - Press Enter key to confirm selection and move to next state.
- setDifficult:
    - Navigate through options with 'w' key to move up and 's' key to move down.
    - Press Enter key for selection.
- playGame:
    - Calling function to initialize game state with all selection above.
    - Transition into play game state.

```
//Waiting for input at current state
switch (currState) {
case mainMenu:
    c = getUart();
    //Enter to choose option
    if (c == '\n') {
        if (arrowPos == helpOption) {
            nextState = helpMenu;
        }
        else if (arrowPos == playOption) {
            nextState = setBackground;
        }
        else if (arrowPos == exitOption) {
            //Clear screen
            clear_screen();
            return;
        }
    }
    //Delete Arrow Position
    if (c == 's') {
        if (arrowPos != exitOption) {
            deleteArrow(300, 300 + arrowPos * 50);
            arrowPos++;
        }
    }
    else if (c == 'w') {
        if (arrowPos != playOption) {
            deleteArrow(300, 300 + arrowPos * 50);
            arrowPos--;
        }
    }
    //Display Arrow Position
    displayArrow(arrow, 300, 300 + arrowPos * 50);
    break;

case helpMenu:
    c = getUart();
    if (c == '\n') {
        nextState = mainMenu;
    }
    displayArrow(arrow, 300, 330 + arrowPos * 50);
    break;
```

```
case setBackground:
    c = getUart();
    if (c == '\n') {
        nextState = setBird;
    }
    else if (c == 'a') { // slide to previous image
        if (current_bg == 0) current_bg = background_LEN - 1;
        else current_bg--;
        setBackgroundStateDisplay();
    }
    else if (c == 'd') { // slide to next image
        if (current_bg == background_LEN - 1) current_bg = 0;
        else current_bg++;
        setBackgroundStateDisplay();
    }
    break;

case setBird:
    c = getUart();
    if (c == '\n') {
        nextState = setDifficult;
    }
    else if (c == 'a') { // slide to previous image
        if (current_bird == 0) current_bird = bird_allArray_LEN - 1;
        else current_bird--;
        setBirdStateDisplay();
    }
    else if (c == 'd') { // slide to next image
        if (current_bird == bird_allArray_LEN - 1) current_bird = 0;
        else current_bird++;
        setBirdStateDisplay();
    }
    break;
```

```
case setDifficult:
    c = getUart();
    //Enter to choose option difficulty
    if (c == '\n') {
        if (arrowPos == easy) {
            difficulty = 1;
        }
        else if (arrowPos == normal) {
            difficulty = 2;
        }
        else if (arrowPos == extreme) {
            difficulty = 3;
        }
        nextState = playGame;
    }
    //Delete Arrow Position
    if (c == 's') {
        if (arrowPos != extreme) {
            deleteArrow(280, 150 + arrowPos * 100);
            arrowPos++;
        }
    }
    else if (c == 'w') {
        if (arrowPos != easy) {
            deleteArrow(280, 150 + arrowPos * 100);
            arrowPos--;
        }
    }
    //Display Arrow Position
    displayArrow(arrow, 280, 150 + arrowPos * 100);
    break;

case playGame:
    // game_run();
    init_round_game();
    nextState = mainMenu;
    break;

default:
    break;
}
```

Figure 46: Switch block to execute game state

k) End game animation with generate color code.

endgameAnimation() function fills the screen slowly from top-left to bottom-right corner with rectangle whose color vary depending on the coordinate. 40x40-pixel sections of the screen are iterated over by two nested loops. It call the generateColor() function to receive the color code base on the current coordinate (x,y). The drawRectARGB32() is used to draw the rectangle on the screen. We use the set_wait_timer to delay the speed of drawing each rectangle.

```
//Display end game animation to
void endgameAnimation() {
    unsigned int currentColor;
    for (int y = 0; y < screenHeight; y += 40) {
        for (int x = 0; x < screenWidth; x += 40) {
            currentColor = generateColor(x, y);
            drawRectARGB32(x, y, x + 40, y + 40, currentColor, 1);
            //Monitor speed of screen cleaning
            set_wait_timer(1, 5);
            set_wait_timer(0, 5);
        }
    }
}
```

Figure 47: endGameAnimation()

generateColor() generates ARGB32 color based on the coordinate (x, y). It returns an ARBG32 color code value.

```
//Produce color base on position x and y
unsigned int generateColor(int x, int y) {
    unsigned char r = (x + y) % 255;
    unsigned char g = x % 255;
    unsigned char b = y % 255;
    unsigned char a = 255; // Full alpha for visibility

    return (a << 24) | (r << 16) | (g << 8) | b; // Return as ARGB32
}
```

Figure 48: generateColor()

l) Game starting and running Logic

Global Game Attributes

- Difficulty: which is selected and set in the menu part
- Game score and current round: setting before and during playing game

```
int game_scores; // The player's score
char game_scores_str[10]; // The player's score in string
int current_round = 1;
```

Figure 49. Global game scores and current round

- Bird, pipes list to store list of obstacles inside each round, and pipes_size represents the number of obstacles inside each round. Because the program cannot use standard library in C language, the pipes list is set to have maximum length to be 100.

```
int pipes_size;
pipe pipes[MAX_PIPES_SIZE];
Bird bird;
```

Figure 50 Bird, pipes, and pipe list length

```
#define EASY_pipes_size 2
#define MEDIUM_pipes_size 5
#define HARD_pipes_size 10
#define MAX_PIPES_SIZE 100
```

Figure 51 Maximum pipe list length and pipe list length based on difficulty rate

Initial Settings For Game

These settings are implemented when the user chooses the option 'playGame' in menu:

```
case playGame:
    // Clear the screen
    clear_screen();
    // Set the difficulty setting
    init_difficulty_setting();
    // init
    game_scores = 0;
    // Display the background
    backgroundDisplay();
    backupRegion(0, 0, screenWidth, screenHeight);
    // Reset the round and characteristics
    reset_round();
    reset_characteristics();
    // Set the current state to the game
    nextState = 0;
    currState = playGame;
    break;
```

Figure 52. playGame option settings

The details of these settings are mentioned below:

- Clear old screen and background using: clear_screen()
- Settings the game based on the chosen difficulty level: init_difficulty_setting()
  This function will define the number of obstacles, and the obstacle moving velocity inside the game based on selected difficulty rate.

```
// Difficulty Utilities
void init_difficulty_setting() {
    switch (difficulty) {
    case 1: // Easy
        // set number of pipes
        pipes_size = EASY_pipes_size;
        // set pipe speed
        pipe_move_speed = 15;
        break;
    case 2: // Medium
        // set number of pipes
        pipes_size = MEDIUM_pipes_size;
        // set pipe speed
        pipe_move_speed = 17;
        break;
    case 3: // Hard
        // set number of pipes
        pipes_size = HARD_pipes_size;
        // set pipe speed
        pipe_move_speed = 25;      Nhung, 17 hours ago via PR #2 • Mod
        break;
    default:
        pipes_size = EASY_pipes_size; // Fallback to a default value
        break;
    }
}
```

Figure 53 init_difficulty_setting()

- Reset game scores
- Draw background and backup the background for restoring later when performing animation in game using background_display() and backup_region()
- Reset current gaming round to be the first round using reset_round():
  This function will revert current round to be 1, in case the program has any problems when regenerating the game after several plays.

```
void reset_round() {
    current_round = 1;
}
```

Figure 54 reset_round()

- Reset other game characteristics to default using reset_characteristics():
  Because each round will have different characteristics. For example, round 1 only has pipe obstacles, while round 2 has balloon obstacles, but round 3 have both balloon obstacles, and the pipes moving both horizontally and vertically. Therefore, to keep the game consistency, when the user re-selects the game playing options, the program calls this function.

```
void reset_characteristics() {
    // reset
    // vertical move: none, balloon: none, pipe: yes
    _is_vertical_move = false;
    _is_have_balloon = false;
    _is_have_pipe = true;
}
```

Figure 55 reset_characteristics()

Initial Settings For Each Round – init_round_game()

- This part will generate settings based on current round. By default and when start playing game, the current round is 1. Firstly, this function converts current game scores to string, and display to game screen. Then, it will Implement to generate the list of obstacles inside each round, and the bird attributes.

```
// Round
void init_round_game() {
    // function to init game for each round
    switch (current_round) {
    case 1:
        // scores
        convert_scores_to_str();
        gamingScoresDisplay();

        init_bird();
        init_pipes();
        // run game
        game_run();
        break;
    case 2:
        // reset
        gameWin = false;
        // scores
        convert_scores_to_str();
        gamingScoresDisplay();      Nhung, yesterday via PR #2 • Done i

        // pipe size
        pipes_size *= 3;
        // set characteristics
        enable_balloon();
        // extreme mode: have both balloon and pipe
        if (difficulty == 1 || difficulty == 2) { // only have balloon
            disable_pipe();
        }

        // setting up game
        init_bird();
        init_pipes();
        // run game
        game_run();
        break;
    case 3:
        // reset
        gameWin = false;
        // scores
        convert_scores_to_str();
        gamingScoresDisplay();
```

Figure 56 ***init_round_game() - 1***

```
        break;
    case 3:
        // reset
        gameWin = false;
        // scores
        convert_scores_to_str();
        gamingScoresDisplay();

        // pipe size
        if (difficulty == 1) {
            pipes_size = EASY_pipes_size;
        }
        // set characteristics
        enable_vertical_move();
        enable_pipe();
        if (difficulty == 3) {
            pipe_move_speed += 5;
        }

        // setting up game
        init_bird();
        init_pipes();
        // run game
        game_run();
        break;
    default:
        break;
    }
}
```

Figure 57 init_round_game() – 2

- **Generate obstacles**. The program generates obstacles list based on the settings of the round and difficulty rate. There are three types: both having balloons and pipes, having only balloons, and having only pipes.

```c
// handle game logics
void init_pipes() {
    // define current offset x
    int current_offset_x = screenWidth / 2;
    // define current pipe index
    current_pipe_index = 0;
    // define display type based on characteristics
    int display_type = 0;
    if (is_have_balloon() && is_have_pipe()) { // both have pipe and balloon
        display_type = 3;
    }
    else if (is_have_pipe()) { // only have pipe
        display_type = 1;
    }
    else { // only have balloon
        display_type = 0;
    }
    for (int i = 0; i < pipes_size; i++) {
        // init pipes x and y
        // init x position
        int pipe_distance = rand_range(PIPE_DISTANCE_MIN, screenWidth / 3); // easy

        // Decide randomly whether to initialize a balloon or a pipe
        if (display_type == 3) { // both have pipe and balloon
            if (rand() % 2 == 0) { // 50% chance to create a balloon
                generate_obstacle(1, i, pipe_distance, &current_offset_x);
            }
            else {
                generate_obstacle(0, i, pipe_distance, &current_offset_x);
            }
        }
        else if (display_type == 1) { // only have pipe
            generate_obstacle(0, i, pipe_distance, &current_offset_x);
        }
        else { // only have balloon
            generate_obstacle(1, i, pipe_distance, &current_offset_x);
        }

    }
}
```

Figure 58 init_pipes()

Besides, there is also a function to generate each obstacle, either pipe or balloon. This function generate_obstacle() will generate based on the required type. If the type is pipe, the program generates the obstacle with top and bottom objects, having random horizontal x and vertical y positions with constant limitations. If the type is balloon, the program generates random y and x positions similarly to the pipe one.

The difference is its type height, and width, and the balloon has random index to choose which image to display as balloon obstacle.

```
void generate_obstacle(int type, int index, int pipe_distance, int* p_current_offset_x) {
    // type: 0 - pipe, 1 - balloon
    if (type == 0) {
        pipes[index].type = 0; // pipe
        pipes[index].top.x = *p_current_offset_x + pipe_distance;
        pipes[index].top.size.width = PIPE_WIDTH;

        // offset for next pipe or balloon
        *p_current_offset_x = pipes[index].top.x + pipes[index].top.size.width;
        pipes[index].bottom.x = pipes[index].top.x;
        pipes[index].bottom.size.width = pipes[index].top.size.width;

        pipes[index].top.y = rand_range(screenHeight - PIPE_GAP_MAX, PIPE_TOP_MIN);  // Random height for top pipe's bottom end
        pipe_gap = rand_range(PIPE_GAP_MAX, PIPE_GAP_MIN);  // Random gap between pipes
        pipes[index].bottom.y = pipes[index].top.y + pipe_gap;

        if (pipes[index].top.x < screenWidth) {
            draw_pipe(pipes[index]);
        }
    }
    else if (type == 1) {
        // generate balloon
        pipes[index].bottom.x = *p_current_offset_x + pipe_distance;
        // pipes[index].bottom.y = screenHeight;
        pipes[index].bottom.y = rand_range(screenHeight / 2, screenHeight + 10);
        pipes[index].bottom.size.width = BALLOON_WIDTH;
        pipes[index].bottom.size.height = 150;

        // offset for next pipe or balloon
        *p_current_offset_x = pipes[index].bottom.x + pipes[index].bottom.size.width;
        pipes[index].type = 1; // balloon
        // random balloon in the image list
        int temp_index = rand_range(0, obstacle_balloon_allArray_LEN - 1);
        pipes[index].display_index = temp_index;

        // if inside the screen, then draw it
        if (pipes[index].bottom.x < screenWidth) {
            draw_balloon(pipes[index]);
        }
    }
}
```

Figure 59 generate_obstacle()

- **Initialize bird**. In this part, the program initializes the bird width, and bird height to be 60. Besides, it also defines the bird position offset to be near the middle of the screen. Finally, the program sets the bird velocity to be 0 and and draw bird to the screen.

```
void init_bird() {        Nhung, 2 days ago via P
    bird_width = 60;
    bird_height = 60;
    // TODO: add round - change bird
    set_bird_position(200, screenHeight / 2);
    bird.vertical_velocity = 0;
    draw_bird(bird, bird_width, bird_height);
}
```

Figure 60 init_bird()

Running Game with Conditions

- After finishing initializing, the program starts running game using game_run() function. To avoid the user feeling confused and does not know why they lose when right after starting game, the teams pauses the game until the user first inputs 'space' character to start game. During running the game, firstly, setting the timer counter first. Then, the program prompts the users to input a character. If that character is 'space' character, the bird flaps. Afterwards, the program generates animation of obstacles,

and bird, but still keep the scores display onto the game screen – in `Game Movements and Animation`. Then, it checks if game is game over, or win. If the state is game over, or winning game on the last round, the program exits game and displays summary scores. If the user wins the game but not on the last round, the program will switch round – mentioned in `Switch Round` section.

```
void game_run() {        hakhanhne, 3 days ago via PR #1 • bird move pipe move added but kinda cringy

    while (_is_resumed == false) {
        char c = uart_getc();
        if (c == ' ' && _is_resumed == false) { // if space is pressed and game is not resumed
            if (current_round != 1) {
                changeRoundRemove();
            }
            resume_game();
            wait_msec(FRAME_MOVING_GAME);
            break;
        }
    }
    // if resume, then start game
    while (1) {
        // set_wait_timer(1, 100);
        // 10: easy - 15: normal - 20: extreme
        set_wait_timer(1, FRAME_MOVING_GAME);
        char play_char = getUart();
        if (play_char == ' ') {
            // bird.vertical_velocity -= 3;       // Gravity pulls the bird down
            // bird.y -= FLAP_STRENGTH;
            flap_bird();
        }
        move_pipes();
        update_bird();

        gamingScoresDisplay();

        if (gameOver) {
            end_game_over_action();
            break;
        }
```

Figure 61 game_run()

```
        if (gameWin) {
            // if win, then check if last round
            // if last round, then end game
            if (check_last_round()) {
                end_game_win_action();
            }
            // if not last round, then change round
            else {
                next_round();
                pause_game();
                changeRoundDisplay();
                init_round_game();
            }
            break;
        }
        // set_wait_timer(0, 100);
        set_wait_timer(0, FRAME_MOVING_GAME);
        // wait_msec(400);
    }

}
```

Figure 62 game_run() – 2

Game Movements and Animation

The bird movement. The team chooses screen height from top to bottom to be vertical axis, and from left to right to be horizontal axis. The gravity of the game world is set to be `9.8`. Assume the bird mass to be `1`. If the user presses nothing, for every time the frame updates, the bird velocity increases by the

gravity. When the user presses `space` character, the bird flaps and moves up vertically with flap strength to be `-32`, whose direction is opposite to be the gravity direction. In other words, the movement of bird is calculated with the equation below:

$$Current\ Bird\ Velocity = gravity * n_1 + v_{flap}$$

- n_1: number of the moved down frames
- If the current bird velocity is positive, the bird is moving down, and vice versa.

This behavior is developed inside update_bird() function. This function firstly clear old bird image displayed on the game screen, then it will validates if the user wins current round or not based on number of passing obstacles. If it wins, it will change game win state and exits the function. Next, the program updates the bird velocity based on provided equation. Afterwards, the game checks if the bird collides with the obstacles, or reaches the borders of the screen, the game state is set to be game over and exits the function. If the bird does not collidate with ceiling or collide with obstacles, the program validates if the bird passes an obstacle to increase current scores and display to the game screen. Finally, the program redraws the bird with updated position.

```
void update_bird() {        hakhanhne, 3 days ago via PR #1 • combine with nhung …
    clear_bird();

    // Win game
    if (current_pipe_index == pipes_size) {
        // win game
        end_game_win();
        // exit
        return;
    }

    bird.vertical_velocity += GRAVITY;  // Gravity pulls the bird down
    bird.y += bird.vertical_velocity;   // Modify y-coordinate by vertical velocity

    // Check for ceiling collision
    if (validate_bird_overflow() || validate_bird_obstacle_collision()) {
        // game over - lose
        end_game_over();
        // exit
        return;
    }
    else if (validate_bird_passing_pipe()) {
        // if pass pipe, then increase score
        current_pipe_index++;
        game_scores++;
        convert_scores_to_str();
        // display changed scores
        clearGameScoresDisplay();
        gamingScoresDisplay();
    }

    draw_bird(bird, bird_width, bird_height);
}
```

Figure 63. update_bird()

The obstacle movement. This function moves the obstacles to the left using linear motion with dynamic speed based on selected difficulty and current round.

For **pipe** obstacle, if the obstacles are inside the screen, clear obstacles in old positions. Then, move obstacles to the left. On the other hand, if the obstacles are outside the screen width horizontally, then not draw them. Besides, if there is a vertical moving characteristic, then move obstacles vertically with constant

number to be `15`. The vertical direction depends on their current y positions. If their current position reaches out of bounds, then the pipes will revert vertical direction.

For **balloon** obstacle, similar to the pipe one, firstly, clear balloons in old positions. Then move balloons to the left horizontally, and to the top vertically. If balloon goes off the screen on top, then resets its position to be at the bottom of the screen height.

Finally, after setting all new positions, redraw the obstacles.

```
void move_pipes() {
    static int vertical_directions[MAX_PIPES_SIZE]; // 0 for up, 1 for down. Initialize all pipes to move up initially.
    static int initialized = 0; // To check whether static variables are initialized or not.
    // Initialize vertical_directions array
    if (!initialized) {
        for (int i = 0; i < pipes_size; i++) vertical_directions[i] = 0; // Initialize all pipes to move up initially.
        initialized = 1;
    }

    for (int index = 0; index < pipes_size; index++) {
        // Moving pipe
        if (pipes[index].type == 0) {
            // clear old
            if (pipes[index].top.x + pipes[index].top.size.width <= screenWidth && pipes[index].top.x > 0) {
                clear_pipe(pipes[index]);
            }
            // Move the pipe to the left
            pipes[index].top.x -= pipe_move_speed;
            pipes[index].bottom.x -= pipe_move_speed;
            // Skip drawing if the pipe is off the screen
            if (pipes[index].top.x + pipes[index].top.size.width <= 0 || pipes[index].top.x <= 0) continue;  // If this pipe skip the screen, skip it
            // Check if the pipe is moving vertically
            if (is_vertical_move()) {
                // Vertical Movement for balloon and pipe having vertical movement
                int movement = vertical_directions[index] == 0 ? -PIPE_VERTICAL_SPEED : PIPE_VERTICAL_SPEED;
                // Move the pipe up or down
                pipes[index].top.y += movement;
                pipes[index].bottom.y += movement;

                // Change direction if hitting vertical limits
                if (pipes[index].top.y <= PIPE_VERTICAL_LIMIT || pipes[index].top.y >= screenHeight - PIPE_VERTICAL_LIMIT - pipe_gap) {
                    vertical_directions[index] = 1 - vertical_directions[index]; // Toggle direction
                }
            }

            // if in the screen, draw it
            if (pipes[index].top.x + pipes[index].top.size.width <= screenWidth) {      You, 20 hours ago via PR #2 • Add balloon and black hole …
                // Display the pipe on the screen at its new position.
                draw_pipe(pipes[index]);
            }
        }
```

Figure 64. move_pipes()

```
        // Moving balloon
        else if (pipes[index].type == 1) {
            // clear balloon
            if (pipes[index].bottom.x <= screenWidth && pipes[index].bottom.x > 0) {
                clear_balloon(pipes[index]);
            }
            // Move the balloon to the left
            pipes[index].bottom.x -= (pipe_move_speed);
            int movement = vertical_directions[index] == 0 ? -BALLOON_RISE_SPEED : BALLOON_RISE_SPEED;
            pipes[index].bottom.y += movement; // Move balloon upwards.

            if (pipes[index].bottom.y < 0) { // If balloon goes off the screen at the top, reset position or remove it
                pipes[index].bottom.y = screenHeight; // Reset balloon position to bottom.
            }
            // redraw
            if (pipes[index].bottom.x + pipes[index].bottom.size.width <= screenWidth && pipes[index].bottom.x > 0) {
                draw_balloon(pipes[index]);
            }
        }

    }
}
```

Figure 65 move_pipes() - 2

Switch Round

To switch round, the team generates reusable utility round functions:

```
// Round Utilities
void reset_round() {
    current_round = 1;
}
void next_round() {
    current_round++;
}
bool check_last_round() {
    // check if last round
    return current_round == 3;
}
```

Figure 66 Utility Round Functions

When there is winning detection in game_run function and if the current round is not last round, there will be a switching round process, including increasing current round, pausing game, and call init_round_game again. When in new play, there is a new round display to game screen to notify which rounds the users are in now.

```
if (gameOver) {
    end_game_over_action();
    break;
}
if (gameWin) {
    // if win, then check if last round
    // if last round, then end game
    if (check_last_round()) {
        end_game_win_action();
    }
    // if not last round, then change round
    else {
        next_round();
        pause_game();
        changeRoundDisplay();
        init_round_game();
    }
    break;
```

Figure 67 Switching Round process

There is a changeRoundDisplay function to display new round to the game screen, with delay function for debouncing.

```
void changeRoundDisplay() {
    backgroundDisplay();
    backupRegion(0, 0, screenWidth, screenHeight);

    // convert round to string
    char round_str[2];
    citoa(current_round, round_str, 10);

    // display round
    drawSentence("Round", 350, 80, changeRoundColorCode);
    drawSentence(round_str, 600, 80, changeRoundColorCode);
    // prevent debounce
    wait_msec(1000);

}
```

Figure 68 changeRoundDisplay

6. Implementation Results

- When in the main menu:

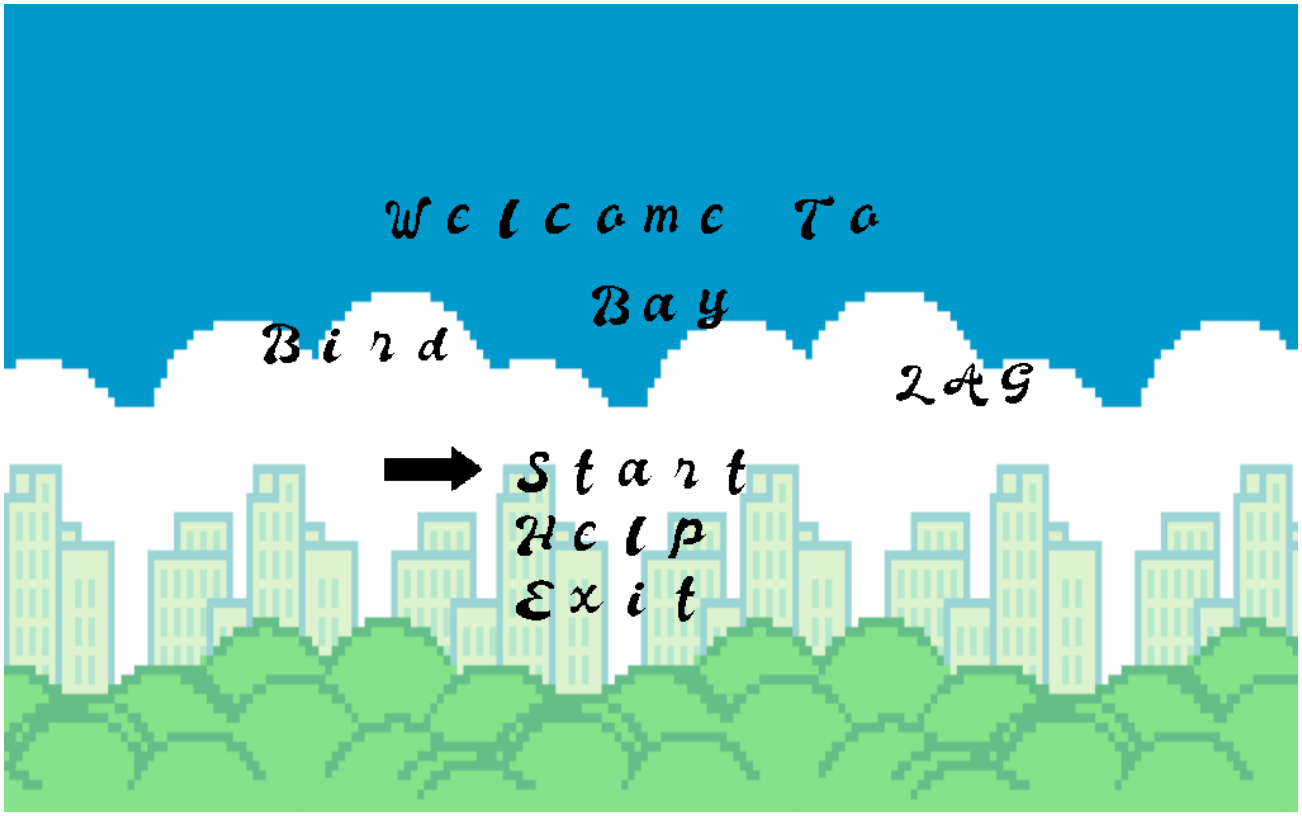

Figure 69 Results - Main Menu

- When in Help screen:

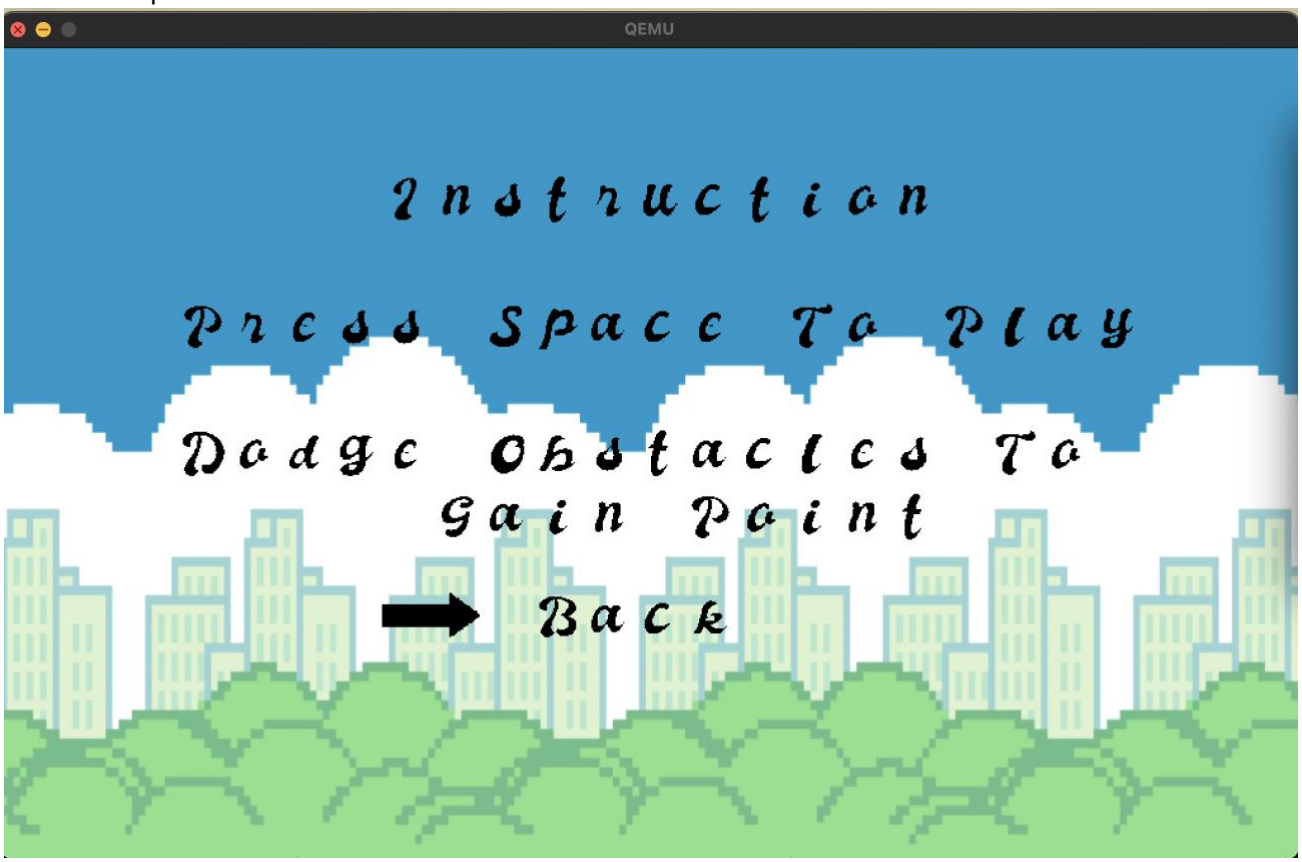

Figure 70 Results - Help screen

- When select options using key `w` and `s` to choose main menu:

Figure 71 Results - select options using key `w` and `s` to choose main menu

- When chooses start option:

Figure 72 Results - Pressing start option

- When pressing `a` and `d` to choose background:

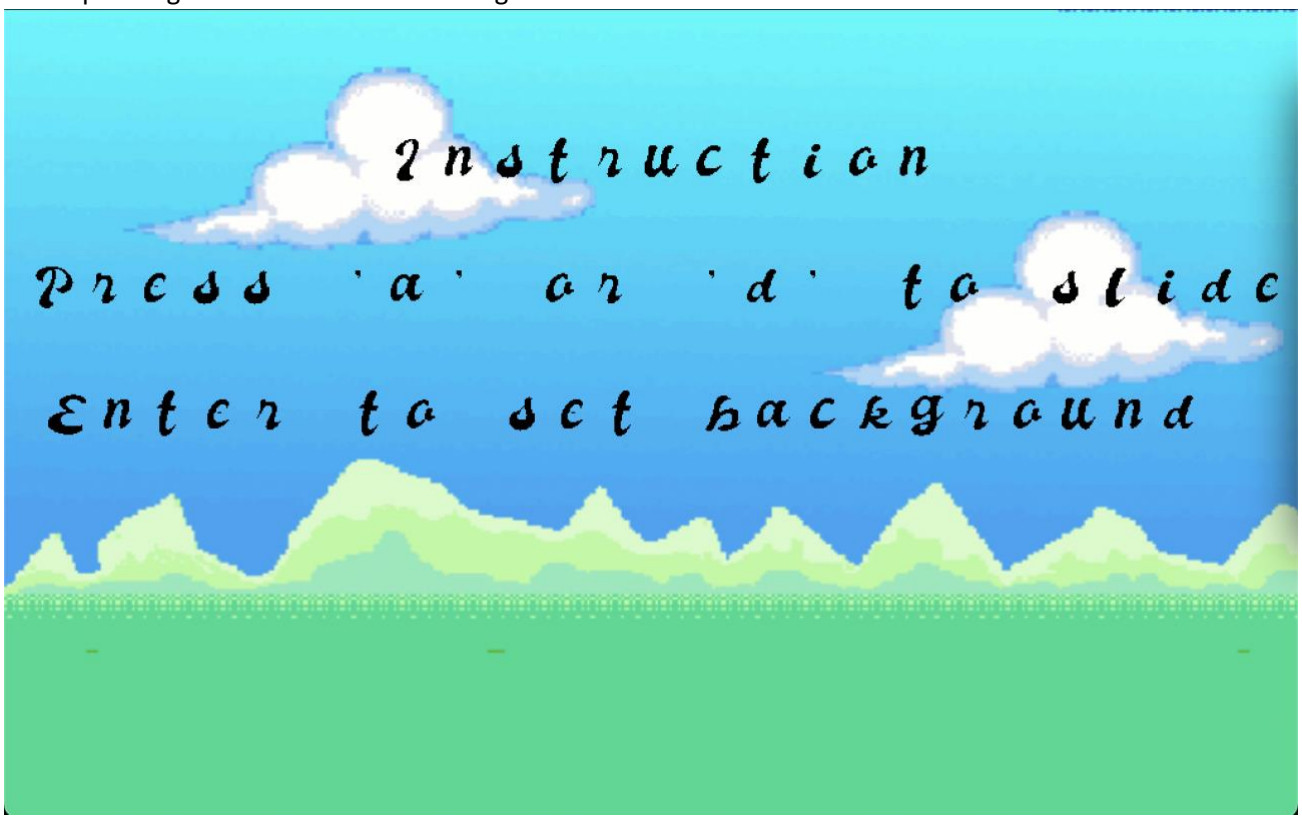

Figure 73. Results - press `a` - `d` to choose background

- After pressing enter, then choose bird skin:

Figure 74. Results - choosing bird skin

- Slide `a` and `d` to select bird:

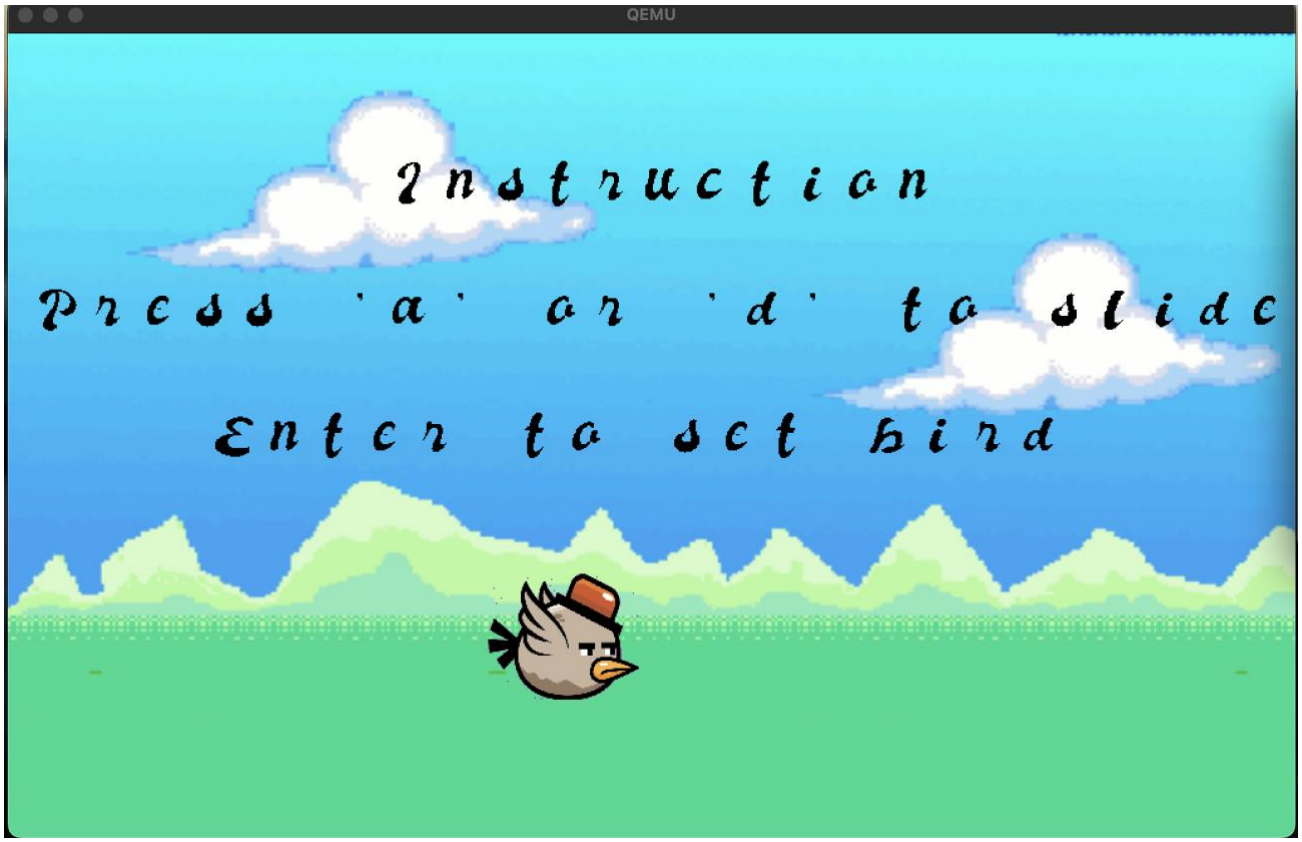

Figure 75. Results - choosing skin

- After choosing bird skin, then choose the difficulty:

Figure 76. Results - Choose difficulty

- During playing game:

Figure 77. Results - Playing game

- During playing game – score 1:

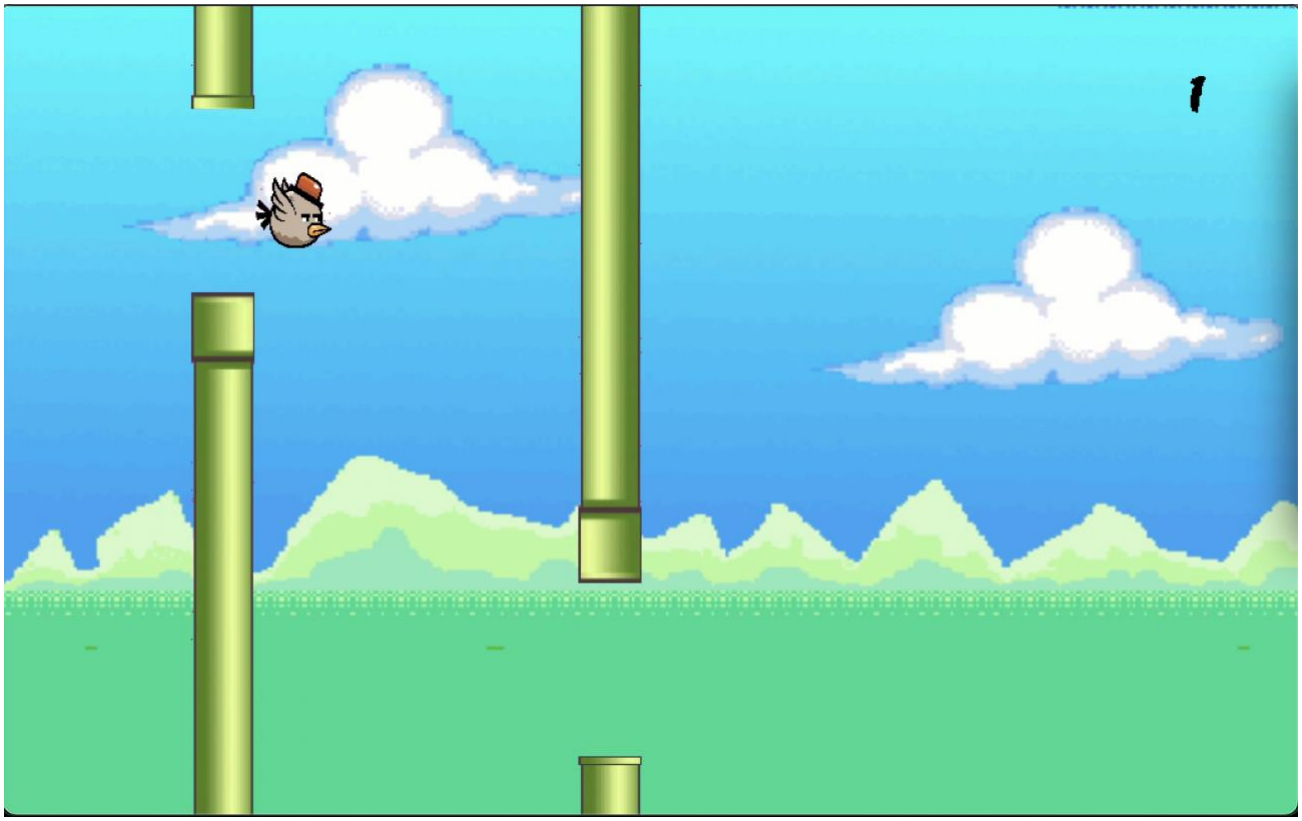

Figure 78 Results - Playing game - scoring 1

- When playing game in round 2:

Figure 79 Results - playing game in round 2

Figure 80 Results - playing game in round 2 – 2

- When playing in game in round 2 – after scoring:

Figure 81 Results - playing game in round 2 - after scoring

- After losing or winning game:

Figure 82 Results - after winning or losing game

- After pressing any key, there is game over display animation:

Figure 83. Results - ending game animation

- Ending game animation after running nearly complete:

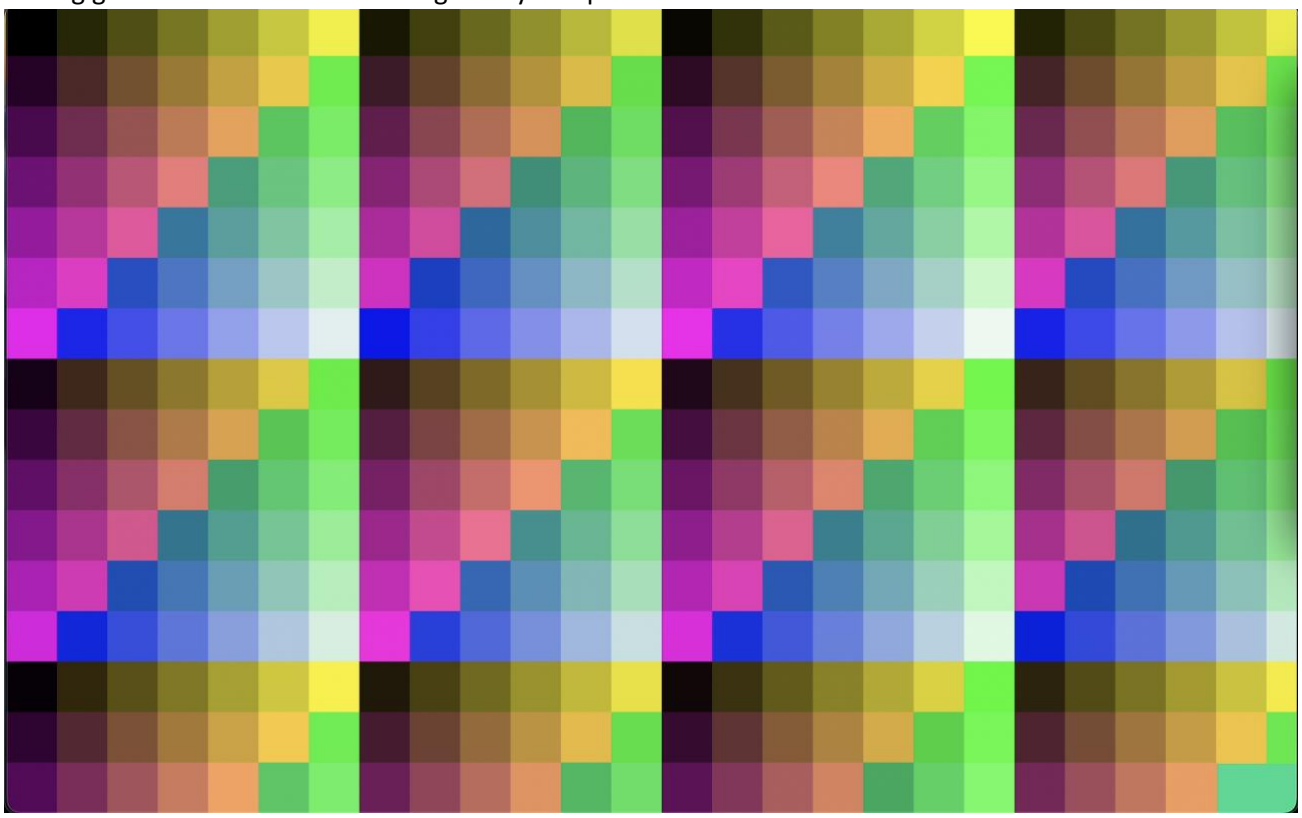

Figure 84. Results - ending game animation – nearly complete

- The game menu after finishing playing:

Figure 85 Results - The game menu after finishing playing

In summary, the game application runs as expected having switching round, different difficulty level, and the user can also view instruction before playing. Finally the game application when running on MacOS and Windows system is laggy as expected and as our name – "Bird Bay Lag".

However, when running on Raspberry Pi 4, the game application becomes laggy over our expected rate. Besides, thanks to this laggy rate, the team can understand how frame buffer generates and draws pixels to the screen.

IV. CONCLUSION

Through our dedication for the project, the end result was a fully functional OS, proficient in displaying crisp images, streaming smooth videos, and showcasing readable text. Moreover, we integrated a game, designed from the ground up, to complement the capabilities of our OS. As this was a sizeable group project, it taught our team the importance of syncing our schedules and enhancing effective communication. Altogether, this assignment was not just about obtaining skills but also about understanding the essence of collaboration and applied what we learnt about Embedded Operating System.

V. REFERENCES (USE IEEE STYLES)

[1] Wikipedia, "Rgba Color Model,". Available: https://en.wikipedia.org/wiki/RGBA_color_model (accessed Sep. 26, 2023).

[2] fontmeme.com, Available: https://fontmeme.com/fonts/static/235021/autolova-font-character-map.png

[3] "image2cpp," javl.github.io. Available: https://javl.github.io/image2cpp/

[4] “FFmpeg”, ffmpeg.org. Available: https://ffmpeg.org/

[5] ThanhNien, “Chàng Trai Viết Game flappy bird Gây Sốt Toàn Cầu,” thanhnien.vn. Available : https://thanhnien.vn/chang-trai-viet-game-flappy-bird-gay-sot-toan-cau-18584583.htm (accessed Sep. 26, 2023).

[6] L. McGregor, “A Beginner’s Guide to Frame Rates ,” https://www.premiumbeat.com/ . Available: https://www.premiumbeat.com/blog/beginners-guide-to-frame-rates/ (accessed Sep. 26, 2023).