VILNIUS UNIVERSITY
ŠIAULIAI ACADEMY

BACHELOR PROGRAMME SOFTWARE ENGINEERING

Programming of Embedded Systems

Flappy Bird Game

Laboratory work No. 5

Student : Hanna Asiadouskaya

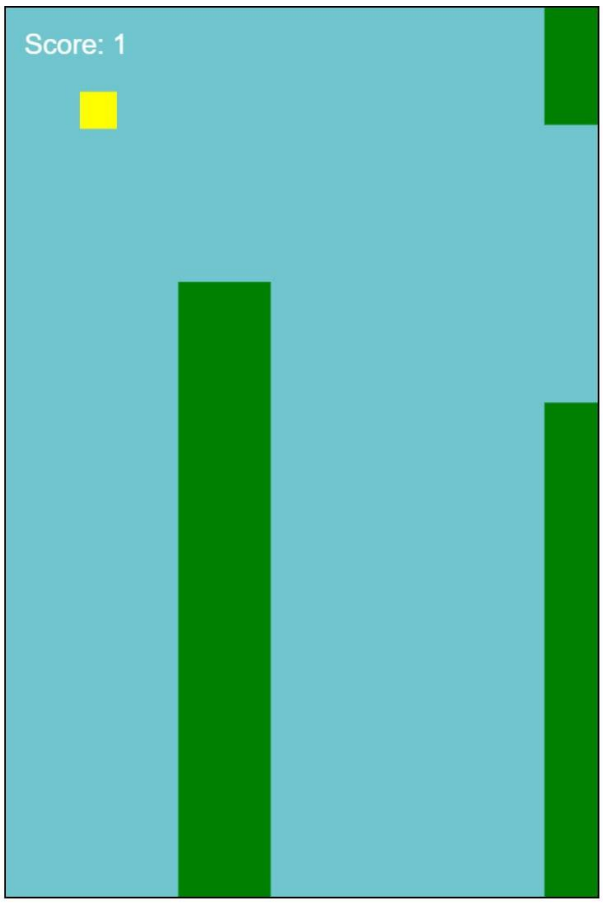
Lecturer: dr. Nerijus Ramanauskas

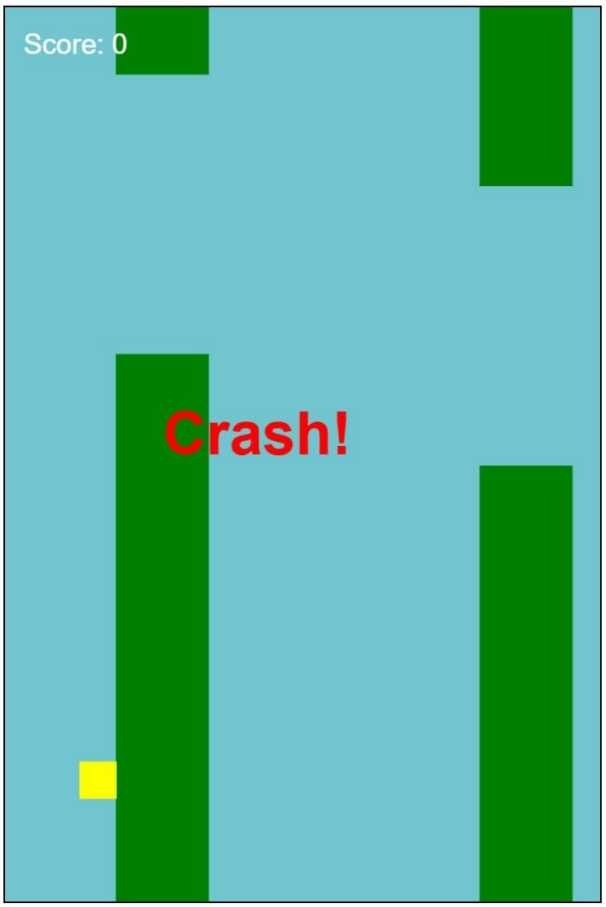
Šiauliai, 2025

Aim of the laboratory work:

The aim of the practical work is to show obtained knowledge in embedded programing using STM32 Nucleo F756ZG microcontroller, by creating game Flappy Bird.

Photos:





Code and Comments:

Imports for network configuration, delays and time, socket to handle HTTP request and import to control LEDs and buttons.

```
1  import network
2  import time
3  from usocket import socket
4  from machine import Pin
```

Configuration to visually indicate game events, green for passing, blue for flapping and red to indicate crash. And indication that default state of the button is inactive(low)

```
6  green = Pin("LED1", Pin.OUT)
7  blue = Pin("LED2", Pin.OUT)
8  red = Pin("LED3", Pin.OUT)
9
10 button = Pin("SW", Pin.IN, Pin.PULL_DOWN)
```

Next we are setting variables. For example, restart_hold_ms meaning the player must hold button for 2 seconds and the game will restart.

```
12 restart_flag = False
13 button_pressed_time = 0
14 restart_hold_ms = 2000
15 flap_flag = False
16 last_button_state = 0
17 last_flap_time = 0
18 flap_debounce_ms = 150
```

Next block is for handling button input.

First is checking is the button pressed. Use global so we can update variables. Check current state of the button and time, so later we can check the time the button pressed.

Writer debounce logic, so if enough time passed from the last flap it can be pressed again. Flag_flap to trigger flap. Blink the blue led to indicate the flap. Update the

last flap time to enforce debounce on the next flap.

```
def check_flap():
    global flap_flag, last_button_state, last_flap_time, restart_flag, button_pressed_time
    current_state = button.value()
    now = time.ticks_ms()

    if current_state == 1 and last_button_state == 0:
        # Button just pressed
        button_pressed_time = now
        if not restart_flag and time.ticks_diff(now, last_flap_time) > flap_debounce_ms:
            flap_flag = True
            blue.high()
            time.sleep_ms(30)
            blue.low()
            last_flap_time = now
```

Second is to handle if button pressed (current state =1 and last button state also 1)

If pressed for more than 2 seconds it will restart and blink. But if you pressed not long enough it will just blink

```
    elif current_state == 1 and last_button_state == 1:
        # Button is held down
        held_time = time.ticks_diff(now, button_pressed_time)
        if held_time > restart_hold_ms:
            restart_flag = True
        elif held_time > 500:
            # Start blinking all LEDs
            if (held_time // 250) % 2 == 0:
                green.on()
                blue.on()
                red.on()
            else:
                green.off()
                blue.off()
                red.off()
```

Button released (current state 0 and last state 1)

No actions – no lights, keeping state as 0

```
51      elif current_state == 0 and last_button_state == 1:
52          # Button released
53          green.off()
54          blue.off()
55          red.off()
56
57      last_button_state = current_state
```

In this function at first we create the view

```
59  def serve_html():
60      html = '''<!DOCTYPE html>
61      <html>
62      <head>
63          <meta charset="UTF-8">
64          <title>Flappy Bird - STM32 Controlled</title>
65          <style>
66              canvas {
67                  display: block;
68                  margin: auto;
69                  background: #70c5ce;
70                  border: 2px solid #000;
71              }
72          </style>
73      </head>
74      <body>
```

Then declaring the size of the canvas for game (surface), and writing script in JS, explaining the actions and gravity.

```
75          <canvas id="gameCanvas" width="640" height="960"></canvas>
76          <script>
```

For example, here bird flapping logic.

From lines 82 to 90 created physics.

Next methods, such as draw – to create yellow bird, flap – if enough cooldown, trigger upwards movements (flap), update – applies gravity and update it position.

From 109 to 116 lines made for prevention of the bird to go off the screen.

```
81          const bird = {
82              x: 100,
83              y: 300,
84              w: 40,
85              h: 40,
86              gravity: 0.08,
87              lift: -5,
88              velocity: 0,
89              lastFlap: 0,
90              flapCooldown: 100,
91
92              draw() {
93                  ctx.fillStyle = "yellow";
94                  ctx.fillRect(this.x - this.w/2, this.y - this.h/2, this.w, this.h);
95              },
96
97              flap() {
98                  const now = Date.now();
99                  if (now - this.lastFlap > this.flapCooldown) {
100                     this.velocity = this.lift;
101                     this.lastFlap = now;
102                 }
103             },
104
105             update() {
106                 this.velocity += this.gravity;
107                 this.y += this.velocity;
108
109                 if (this.y + this.h/2 > canvas.height) {
110                     this.y = canvas.height - this.h/2;
111                     this.velocity = 0;
112                 }
113                 if (this.y - this.h/2 < 0) {
114                     this.y = this.h/2;
115                     this.velocity = 0;
116                 }
```

Next Pipe logic.

First declaring the size, and spawn interval of the pipe. Method update is created to spawn pipe at a random vertical offset every 2 seconds. From line 138 to 150 written logic to move pipes left, so eventually they would be off screen, and green LED indicator when the bird passes the pipe. Draw – to draw pipe from top to button and leaving gap.

```
120         const pipes = {
121             position: [],
122             w: 100,
123             h: 500,
124             gap: 300,
125             dx: 1.2,
126             maxYPos: -300,
127             lastSpawn: 0,
128             spawnInterval: 2000,
129 
130             update() {
131                 const now = Date.now();
132                 if (now - this.lastSpawn > this.spawnInterval) {
133                     this.lastSpawn = now;
134                     const y = this.maxYPos * (Math.random() + 1);
135                     this.position.push({ x: canvas.width, y: y, passed: false });
136                 }
137 
138                 for (let i = 0; i < this.position.length; i++) {
139                     const p = this.position[i];
140                     p.x -= this.dx;
141 
142                     if (!p.passed && p.x + this.w < bird.x) {
143                         p.passed = true;
144                         fetch("/passed");
145                         score++;
146                     }
147 
148                     if (p.x + this.w <= 0) this.position.shift();
149                 }
150             },
151 
152             draw() {
153                 for (const p of this.position) {
154                     ctx.fillStyle = "green";
155                     ctx.fillRect(p.x, p.y, this.w, this.h);
156                     ctx.fillRect(p.x, p.y + this.h + this.gap, this.w, canvas.height);
157                 }
158             }
159         };
```

Next is collision detection. Logic to check if the bird crashed, and if yes then activate red LED

```
161         function checkCollision() {
162             for (const p of pipes.position) {
163                 const pipeTop = p.y + pipes.h;
164                 const pipeBottom = p.y + pipes.h + pipes.gap;
165                 const birdTop = bird.y - bird.h/2;
166                 const birdBottom = bird.y + bird.h/2;
167                 const birdLeft = bird.x - bird.w/2;
168                 const birdRight = bird.x + bird.w/2;
169                 const pipeLeft = p.x;
170                 const pipeRight = p.x + pipes.w;
171 
172                 if (
173                     birdRight > pipeLeft && birdLeft < pipeRight &&
174                     (birdTop < pipeTop || birdBottom > pipeBottom)
175                 ) {
176                     fetch("/crash");
177                     return true;
178                 }
179             }
180             return false;
181         }
```

Function to reset bird, pipes, score and crash state

```
183         function resetGame() {
184             bird.y = 300;
185             bird.velocity = 0;
186             pipes.position = [];
187             score = 0;
188             crashed = false;
189             pipes.lastSpawn = Date.now();
190         }
```

Function that activates to draw everything on the screen, and if crashed to show text “Crash!”

```
192         function draw() {
193             ctx.fillStyle = "#70c5ce";
194             ctx.fillRect(0, 0, canvas.width, canvas.height);
195             pipes.draw();
196             bird.draw();
197             ctx.fillStyle = "white";
198             ctx.font = "30px Arial";
199             ctx.fillText("Score: " + score, 20, 50);
200 
201             if (crashed) {
202                 ctx.fillStyle = "red";
203                 ctx.font = "bold 64px Arial";
204                 ctx.fillText("Crash!", canvas.width / 2 - 150, canvas.height / 2);
205             }
206         }
```

Here is the check of the state, if its crashed or not, update of the pipes and bird. Generally function to keep the game going. Using the requestAnimationFrame() method that tells the browser that you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint.

```
208         function loop() {
209             if (!crashed) {
210                 bird.update();
211                 pipes.update();
212                 if (checkCollision()) {
213                     crashed = true;
214                     setTimeout(resetGame, 1000);
215                 }
216             }
217             draw();
218             requestAnimationFrame(loop);
219         }
```

Here we poll the server and if the command flap it's doing the function flap, and if the command restart we doing the function resetGame. For all of that we set interval 30ms

```
221         // Polling the server for flap events
222         setInterval(() => {
223           fetch("/flap").then(r => r.text()).then(text => {
224             const command = text.trim();
225             if (command === "flap") {
226               bird.flap();
227             } else if (command === "restart") {
228               resetGame();
229             }
230           });
231         }, 30);
```

And calling this function to start the game loop

```
233         loop();
```

Next is network setup. Declare the use of two global variables. Then create a LAN (Ethernet) network interface using MicroPython's network module. Activate the network interface so we configure and use it. Then manually set the IP configuration – static IP of the board, subnet mask, gateway and google's public DNS.

Delay to stabilize network settings. (the last line for debugging)

```
239 def main():
240     global flap_flag
241     global restart_flag
242     nic = network.LAN()
243     nic.active(True)
244     nic.ifconfig(('192.168.1.123', '255.255.255.0', '192.168.1.1', '8.8.8.8'))
245     time.sleep_ms(500)
246     print("Network:", nic.ifconfig())
```

Then socket setup. First, we create TCP socket, then bind the socket to all interfaces on port 80 and then put the socket into listening mode to accept incoming connection.

```
248     s = socket()
249     s.bind(('0.0.0.0', 80))
250     s.listen(1)
```

An infinite loop to continually accept and handle client requests. conn – new socket to talk to the client, addr – client's IP and port, S.accept – waiting for client to connect. Before processing the request, it checks the button state (so it will set flap flag or restart flag). Reads up to 1024 bytes of data from the client, and then decodes it from bytes to UTF-8

```
252     while True:
253         try:
254             conn, addr = s.accept()
255             check_flap()
256             request = conn.recv(1024).decode()
257
```

Next is request handling.

1. Flap request. Checking for flap or restart, if no input then we send empty string. Then send standard HTTP 200 OK response
2. Passed request. JS sends when the bird passes the pipe and the green light is blinking. Sends a standard HTTP 200 OK response
3. Crash request. Triggered when a crash is detected in the game, flashes red and sends a standard HTTP 200 OK response
4. If non of the above matched, calling serve_html() which returns the complete HTML and JS, then sends a standard HTTP 200 OK response
5. Close connection after responding

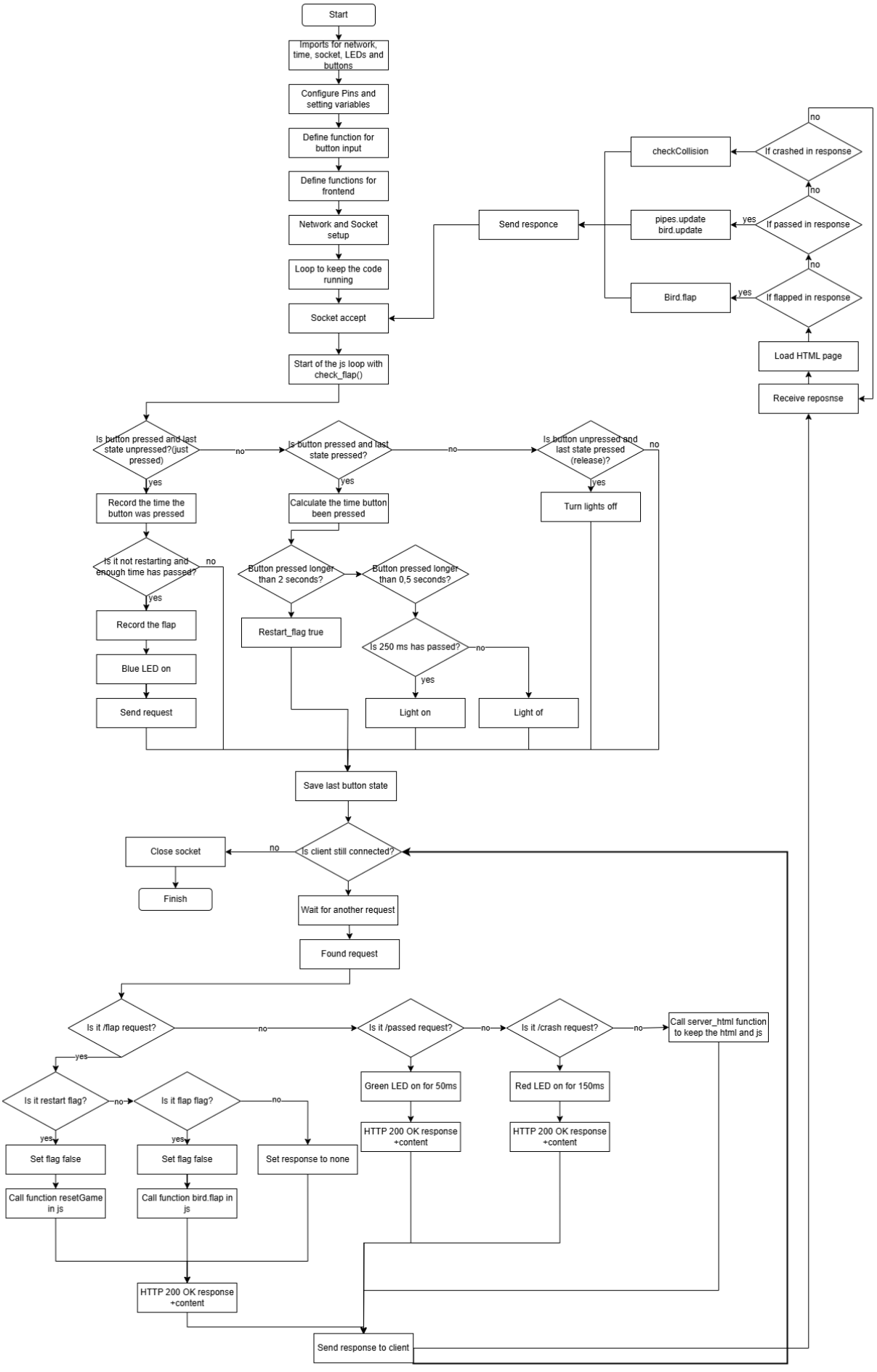
```
258             if "GET /flap" in request:
259                 if restart_flag:
260                     restart_flag = False
261                     response = "restart"
262                 elif flap_flag:
263                     flap_flag = False
264                     response = "flap"
265                 else:
266                     response = ""
267                 conn.send("HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\n" + response)
268 
269             elif "GET /passed" in request:
270                 green.on()
271                 time.sleep_ms(50)
272                 green.off()
273                 conn.send("HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\nok")
274 
275             elif "GET /crash" in request:
276                 red.on()
277                 time.sleep_ms(150)
278                 red.off()
279                 conn.send("HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\nok")
280 
281             else:
282                 html = serve_html()
283                 conn.send("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n" + html)
284 
285             conn.close()
```

Exception handler, so in case of unexpected error prints Error and the reason. Then ensure that the socket is closed even if an error occurred.

And the last thing – entry point

```
287         except Exception as e:
288             print("Error:", e)
289             try:
290                 conn.close()
291             except:
292                 pass
293 
294 if __name__ == "__main__":
295     main()
```

Algorithm:


Start
Imports for network, time, socket, LEDs and buttons
Configure Pins and setting variables
Define function for button input
Define functions for frontend
Network and Socket setup
Loop to keep the code running
Socket accept
Start of the js loop with check_flap()
Send responce
checkCollision
pipes.update bird.update
Bird.flap
If crashed in response
If passed in response
If flapped in response
no
yes
Load HTML page
Receive reposnse
Is button pressed and last state unpressed?(just pressed)
Is button pressed and last state pressed?
Is button unpressed and last state pressed (release)?
Record the time the button was pressed
Calculate the time button been pressed
Turn lights off
Is it not restarting and enough time has passed?
Button pressed longer than 2 seconds?
Button pressed longer than 0,5 seconds?
Record the flap
Blue LED on
Send request
Restart_flag true
Is 250 ms has passed?
Light on
Light of
Save last button state
Is client still connected?
Close socket
Finish
Wait for another request
Found request
Is it /flap request?
Is it /passed request?
Is it /crash request?
Call server_html function to keep the html and js
Is it restart flag?
Is it flap flag?
Green LED on for 50ms
Red LED on for 150ms
Set flag false
Set flag false
Set response to none
HTTP 200 OK response +content
HTTP 200 OK response +content
Call function resetGame in js
Call function bird.flap in js
HTTP 200 OK response +content
Send response to client

Conclusion:

In this study, we successfully practiced programming on the STM32 Nucleo F756ZG microcontroller, showing obtained knowledge during this course, such as Control output, Read input, Control pwm, Send receive data, by developing game Flappy Bird