GEORGIA STATE UNIVERSITY

Department of Computer Science

CSC 3210 Computer Organization Programming

Lab 4: Pong on Game Boy Advance

Suryaprakash Murugavvel

Submitted: November 15, 2024

Contents

1 Introduction

The main goal of this lab is to simplify Pong, one of the first and most famous computer games, for the Game Boy Advance (GBA). Throughout this project, we explore important embedded systems programming ideas, including how to handle I/O interrupts, control sprites, and apply real-time game logic.

The lab's main objective is to develop a playable version of Pong using just one paddle, one ball, and a scoreboard. Basic gameplay features include bouncing the ball off-screen edges, moving the paddle using directional buttons, and updating the score when the paddle strikes the ball. The game is over if the ball hits the screen's lower edge.

The following tasks were assigned to achieve it:

- Setting up a GBA development environment using tools like the Butano engine and an emulator (e.g., mGBA).
- Designing and programming the game components, including sprites and collision logic, in C++.
- Testing the game on an emulator to ensure smooth gameplay and correct scoring mechanics.

This lab emphasizes responsiveness and optimization in real-time programming, offering practical experience in developing an interactive application for systems with limited resources.

2 Appartus

The following tools and resources were utilized in the completion of this lab:

- MacBook M3 Pro running macOS Sequoia 15.0.1
- GNU Compiler on C++ through Terminal[3]
- **Development Tools:**
  - Butano Engine: A high-level modern C++ engine for GBA development.[3]
  - DevkitARM: A development toolkit required for GBA programming.[3]
- Emulator: mGBA (`https://mgba.io/`) for testing the Pong game.
- C++ on VS-Code with Butano-specific libraries for GBA. [4][5][6]
- Source code for pong game (`https://github.com/EHowardHill/butano-pong`)[1]
- Reference Game Online Pong game for understanding mechanics (`https://www.ponggame.org/`).[2]

3 Methods

The lab was divided into two main tasks: setting up the GBA development environment and implementing the Pong game. The steps below outline the approach taken to complete each task.

3.1 Task 1: Setting Up the GBA Development Environment

To begin developing for the Game Boy Advance (GBA), a development environment was configured using the following steps:

1. **Installing Development Tools:**
    - The **DevkitARM** toolkit was downloaded and installed by following the instructions available at `https://devkitpro.org/`. This toolkit provides essential tools for GBA development, including a cross-compiler and libraries.
    - The **Butano Engine**, a high-level C++ engine for GBA, was installed using the documentation provided at `https://gvaliente.github.io/butano/getting_started.html`. The Butano Engine simplifies GBA game development with an intuitive interface and pre-built functionality.
2. **Installing a GBA Emulator:**
    - The **mGBA** emulator (`https://mgba.io/`) was installed to test the developed Pong game. This emulator supports accurate simulation of GBA hardware.
3. **Verifying Setup:**
    - A sample program from the Butano examples library was compiled and executed on the mGBA emulator to confirm the correctness of the setup.

This task ensured that the necessary tools and environment were correctly installed and operational for the subsequent implementation of the Pong game.

3.2 Task 2: Implementing Pong on GBA

Once the development environment was set up, the Pong game was implemented step-by-step as follows:

1. **Game Design:**

   - A background image was loaded, and two sprites were created: a paddle and a ball. The paddle was positioned near the bottom of the screen, while the ball was initially placed at the center.
   - The paddle was set to move horizontally using the GBA's directional buttons, constrained within the screen's boundaries.
   - A scoreboard was implemented to display the player's score, which increments whenever the ball bounces off the paddle.

2. **Ball Movement and Physics:**

   - The ball's initial movement was randomized in both horizontal (`delta_x`) and vertical (`delta_y`) directions. However, the ball always started moving upwards (`delta_y < 0`).
   - Collision detection logic was implemented:
     - The ball bounces off the screen's top left and right edges by reversing its direction.
     - When the ball collides with the paddle, its vertical direction is reversed, and its horizontal speed is adjusted based on the hit position on the paddle.
     - The game ends if the ball reaches the bottom of the screen.

3. **Gameplay Features:**

- The game begins when the player presses the **X** button. The paddle is moved using the left and right buttons, and the game can be restarted by pressing **S** after a game-over event.
- A dotted trace effect was added to visualize the ball's movement path, limiting the number of dots displayed to prevent overflow.
- Sound effects were integrated for collisions and game events using Butano's sound library.

4. **Testing and Debugging:**

   - The game was compiled and run on the mGBA emulator to verify functionality.
   - Iterative adjustments were made to ensure accurate collision detection, smooth gameplay, and responsive controls.

This task resulted in a fully functional Pong game with interactive gameplay, a scoring system, and real-time collision detection, all implemented within the constraints of the GBA hardware.

4 Results and Discussion

Results

The mGBA emulator has been successfully implemented to create and test the Pong game. All necessary elements, such as paddle movement, ball-bouncing physics, collision detection, and a working scoring system, were included in the finished game. Below is a summary of the main results:

- **Paddle Movement:** In response to the directional buttons, the paddle moved smoothly while staying inside the horizontal limits of the screen.
- **Ball Physics:** The ball moved dynamically, precisely bouncing off the paddle and screen edges.
- **Collision Detection:** The ball's proper interaction with the paddle and the screen edges was made possible via accurate collision detection.
- **Scoreboard:** The score was updated reliably every time the ball bounced off the paddle; during game-over occurrences, it was reset.
- **Game-over Mechanic:** The game ended nicely when the ball struck the bottom of the screen, and the "Game Over" notification appeared.
- **Visual Effect:** A trail graphically represented the ball's shifting route, improving the game's visual appeal.
- **Sound Effects:** The gaming experience was further enhanced by adding sound effects triggered by collisions and game events.

A screenshot of the game running on the emulator is provided below.

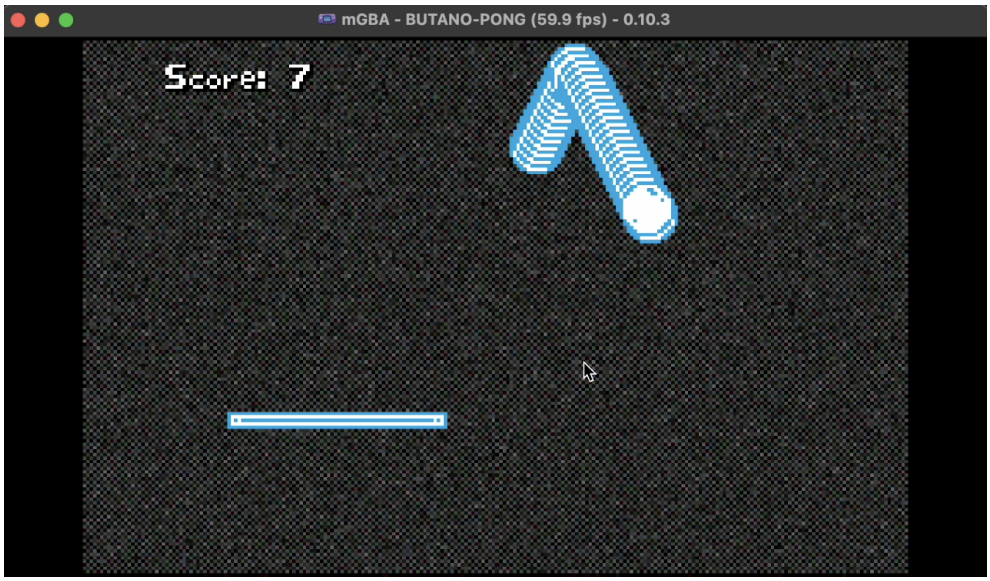

Figure 1: Pong game running on mGBA emulator.

Discussion

Programming for embedded devices and game creation were both greatly enhanced by the GBA platform's implementation of Pong. Below is a discussion of the project's key elements:

- **Collision Detection:** One of the most significant obstacles was accurately identifying ball-paddle collisions. The gameplay was made more challenging by dynamically adjusting the ball's horizontal speed based on its relative position on the paddle. Realistic ball behavior was guaranteed by this method.
- **Optimization for Embedded Systems:** Due to the GBA's limited resources, memory and performance must be carefully optimized. For example:
  - The dotted trace effect was implemented with a size limit to avoid memory overflow.
  - Lightweight sound effects and sprite animations were used to maintain performance.

- **Gameplay Balance:** The ball's randomized horizontal speed added variation, but the maximum speed had to be limited to keep the gameplay balanced and allow players to respond quickly.
- **User Interface:** The user interface was made more evident using text sprites to show the score and game-over notifications. The user experience was greatly improved by adding a restart capability.
- **Testing and Debugging:** Gameplay mechanics were refined through iterative testing on the mGBA emulator. Debugging was used to find and fix problems such as paddle boundary violations and inaccurate collision reactions.
- **Learning Outcomes:** The lab reinforced concepts like resource management, interrupt handling, and real-time interaction. It also emphasized the importance of testing and iterative development in producing a workable embedded system application.

While the game achieved its functional requirements, potential enhancements include:

- Introducing difficulty levels by increasing ball speed as the score rises.
- Adding visual effects or animations to highlight collisions.
- Implementing a high-score system stored in SRAM for long-term tracking.

Overall, the lab showed that it is possible to balance creativity and technological limitations while creating interactive games on embedded devices.

5 Conclusion

This lab gave a deep hands-on experience with embedded system programming by creating a Game Boy Advance (GBA) Pong game. It concentrated on solving the problems caused by hardware limitations, which necessitated effective control of system resources like memory and processing speed. Implementing basic game features like collision detection, sprite movement, and user interaction, as well as setting up the GBA development environment, highlighted how crucial it is to optimize code to work within the constraints of embedded devices. Additionally, the project improved the participants' ability to operate in restricted situations, comprehend how hardware and software interact, and create interactive game logic. In addition to reinforcing basic programming skills for embedded systems, this lab also established a strong basis for future work on more challenging projects and applications in embedded systems development.

6 References

[1] [2] [3] [4] [5] [6]

References

[1] EHowardHill. Source code for pong game and images provided through Github.

[2] Pong Game. https://www.ponggame.org/#google_vignette.

[3] Compile C++ for GBA in under an hour! (using Butano + devkitPro + WSL2). https://www.youtube.com/watch?v=EMeie_gSgDU.

[4] C++ Simple Pong Game (Part 1). https://www.youtube.com/watch?v=y8QL62SDlcQ.

[5] C++ Simple Pong Game (Part 2). https://www.youtube.com/watch?v=soqGGnxK92c&t=512s.

[6] C++ Simple Pong Game (Part 3). https://www.youtube.com/watch?v=Z6hUxXCzKYE.

7 Appendix: C++ Code

```
#include "bn_core.h"
#include "bn_log.h"
#include "bn_sram.h"
#include "bn_music.h"
#include "bn_music_actions.h"
#include "bn_music_items.h"
#include "bn_sound_items.h"
#include "bn_math.h"
#include "bn_string.h"
#include "bn_keypad.h"
#include "bn_display.h"
#include "bn_random.h"
#include "bn_regular_bg_ptr.h"
#include "bn_sprite_text_generator.h"
#include "bn_sprite_animate_actions.h"
#include "bn_sprite_palette_ptr.h"
#include "common_info.h"
#include "common_variable_8x8_sprite_font.h"
#include "bn_sprite_items_paddle.h"
#include "bn_sprite_items_ball.h"
#include "bn_regular_bg_items_bg.h"

int main()
{
    bn::core::init();

    // Creates the background and sprites
    bn::regular_bg_ptr bg = bn::regular_bg_items::bg.create_bg(0, 0);
    bn::sprite_ptr paddle = bn::sprite_items::paddle.create_sprite(0, 60)
        ; // Moved paddle up slightly
```

```
30      paddle.set_rotation_angle(90);
31
32      bn::sprite_ptr ball = bn::sprite_items::ball.create_sprite(0, 0);
33
34      // Initialize the game variables
35      int score = 0;
36      bn::fixed delta_x = 0;
37      bn::fixed delta_y = 0;
38      bool game_over = false;
39      bn::random random;
40
41      // Constants
42      const int PADDLE_Y = 60;  // Fixed paddle Y position
43      const int PADDLE_WIDTH = 32;  // Half width of paddle
44      const int BALL_SIZE = 4;
45      const int SCREEN_TOP = -70;
46      const int SCREEN_LEFT = -120;
47      const int SCREEN_RIGHT = 120;
48      const int SCREEN_BOTTOM = 70;
49
50      // Setup the text display
51      bn::sprite_text_generator text_generator(common::
            variable_8x8_sprite_font);
52      bn::vector<bn::sprite_ptr, 16> text_sprites;
53      text_generator.generate(-6 * 16, -68, "(Press X to start)",
            text_sprites);
54
55      // Vector in order to hold trace
56      bn::vector<bn::sprite_ptr, 32> trace_sprites;
57
58      while(true)
59      {
```

```
60          // Paddle movement logic
61          if(bn::keypad::left_held() && paddle.x() > SCREEN_LEFT)
62          {
63              paddle.set_x(paddle.x() - 2);
64          }
65          else if(bn::keypad::right_held() && paddle.x() < SCREEN_RIGHT)
66          {
67              paddle.set_x(paddle.x() + 2);
68          }
69
70          // Game over -> restart logic
71          if(game_over && bn::keypad::r_pressed())
72          {
73              ball.set_position(0, 0);
74              paddle.set_position(0, PADDLE_Y);
75              delta_x = 0;
76              delta_y = 0;
77              score = 0;
78              game_over = false;
79
80              text_sprites.clear();
81              text_generator.generate(-6 * 16, -68, "(Press X to start)",
                    text_sprites);
82          }
83
84          // Start the game
85          if(bn::keypad::a_pressed() && delta_x == 0 && delta_y == 0 && !
                game_over)
86          {
87              text_sprites.clear();
88              bn::string<32> txt_score = "Score: " + bn::to_string<32>(
                    score);
```

```
89              text_generator.generate(-6 * 16, -68, txt_score, text_sprites
                    );
90
91              // Initialize ball movement with random direction but always
                    moving up initially
92              delta_x = (random.get_int() % 5) - 2;
93              delta_y = -2; // Start moving up
94
95              bn::sound_items::pong.play();
96          }
97
98          // Ball's movement and collision logic
99          if(delta_x != 0 || delta_y != 0)
100         {
101             // Add a new dotted point to the trace at current ball
                    position
102             if (trace_sprites.size() >= 32)  // Prevents overflow by
                    limiting trace length
103             {
104                 // Shifts elements to the front
105                 for(int i = 1; i < trace_sprites.size(); ++i)
106                 {
107                     trace_sprites[i - 1] = trace_sprites[i];
108                 }
109                 trace_sprites.pop_back();
110             }
111
112             // Adds new trace point
113             trace_sprites.push_back(bn::sprite_items::ball.create_sprite(
                    ball.x(), ball.y()));
114
115             // Updates the ball's position
```

```
116             ball.set_x(ball.x() + delta_x);
117             ball.set_y(ball.y() + delta_y);
118
119             // Top wall collision logic
120             if(ball.y() <= SCREEN_TOP)
121             {
122                 ball.set_y(SCREEN_TOP);
123                 delta_y = -delta_y;
124                 bn::sound_items::pong.play();
125             }
126
127             // Side wall collisions logic
128             if(ball.x() <= SCREEN_LEFT)
129             {
130                 ball.set_x(SCREEN_LEFT);
131                 delta_x = -delta_x;
132                 bn::sound_items::pong.play();
133             }
134             else if(ball.x() >= SCREEN_RIGHT)
135             {
136                 ball.set_x(SCREEN_RIGHT);
137                 delta_x = -delta_x;
138                 bn::sound_items::pong.play();
139             }
140
141             // Paddle collision logic
142             if(ball.y() + BALL_SIZE / 2 >= paddle.y() - PADDLE_WIDTH / 2
                    && ball.y() - BALL_SIZE / 2 <= paddle.y() + PADDLE_WIDTH
                    / 2)
143             {
144                 // Checks if ball is within paddle's width
```

```
145                 if(ball.x() >= paddle.x() - PADDLE_WIDTH && ball.x() <=
                        paddle.x() + PADDLE_WIDTH)
146                 {
147                     // Bounces the ball off the top of the paddle
148                     ball.set_y(paddle.y() - (PADDLE_WIDTH / 2) - (
                            BALL_SIZE / 2));
149                     delta_y = -delta_y;
150
151                     // Calculates where on the paddle the ball hit
152                     bn::fixed hit_position = (ball.x() - paddle.x()) /
                            PADDLE_WIDTH;
153
154                     // Adjust the horizontal speed based on where the
                            ball hits the paddle
155                     delta_x = hit_position * 3;
156
157                     score++;
158                     text_sprites.clear();
159                     bn::string<32> txt_score = "Score: " + bn::to_string
                            <32>(score);
160                     text_generator.generate(-6 * 16, -68, txt_score,
                            text_sprites);
161
162                     bn::sound_items::pong.play();
163
164                     // Resets the trace after ball hits the paddle
165                     trace_sprites.clear();
166                 }
167             }
168
169             // Ball goes past paddle -> game over
170             if(ball.y() >= SCREEN_BOTTOM && !game_over)
```

```
171             {
172                 game_over = true;
173                 ball.set_position(0, 0);
174                 paddle.set_position(0, PADDLE_Y);
175                 delta_x = 0;
176                 delta_y = 0;
177                 score = 0;
178 
179                 text_sprites.clear();
180                 text_generator.generate(-6 * 16, -68, "Game Over! Press S
                        to restart", text_sprites);
181 
182                 // Clears trace on game over
183                 trace_sprites.clear();
184             }
185         }
186 
187         bn::core::update();
188     }
189 }
```

```
File: main.tex
Encoding: utf8
Sum count: 1397
Words in text: 1364
Words in headers: 27
Words outside text (captions, etc.): 6
Number of headers: 11
Number of floats/tables/figures: 1
Number of math inlines: 0
Number of math displayed: 0
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  181+1+0 (1/0/0/0) Section: Introduction
  85+1+0 (1/0/0/0) Section: Appartus
  30+1+0 (1/0/0/0) Section: Methods
  150+8+0 (1/0/0/0) Subsection: Task 1: Setting Up the GBA Development Environment
  301+6+0 (1/0/0/0) Subsection: Task 2: Implementing Pong on GBA
  0+3+0 (1/0/0/0) Section: Results and Discussion
  185+1+6 (1/1/0/0) Subsection: Results
  293+1+0 (1/0/0/0) Subsection: Discussion
  132+1+0 (1/0/0/0) Section: Conclusion
  6+1+0 (1/0/0/0) Section: References
  1+3+0 (1/0/0/0) Section: Appendix: C++ Code


(errors:1)
```