University of Northumbria
CM0506 - Small Embedded Systems
-
Assignment 1: Pong

M J Brockway

November 2, 2015

1 Introduction

This assignment is in two parts:

1. an exercise in programming the ARM microprocessor and associated peripherals in C to implement a simple game using digial and analogue inputs and the LCD screen. *You may do this with a partner, or individually.* This program will be assessed on:
   - how well it works - your code will be tested;
   - the structure, style and layout of the code;
2. a short report on relevant aspects of embedded system programming, around 7-8 pages (about 4500 words) in length.

   There is no fixed penalty for exceeding this limit, but a verbose, poorly constructed report containing irrelevant material will not attract a good mark.

1.1 Assessment and Deadlines

The mark for this assignment will contribute 50% to the final mark for this Module. The program and the report will contribute half the marks each.

An indication of how marks are to be allocated is given in sections 2.5 and 3. Important dates for this assignment are:

- Assignment is set in week 7
- Working programs to be demonstrated between 09:00 and 14:00 on Tu**Tuesday 12 January**, 2016. A timetable of demonstrations will be published in advance of the demonstration week.
- Report hand-in on or before **Friday 15 January**, 2016.

- Submission is **electronic**, via the e-Learning Portal.
- Marks and feedback will be available about three weeks thereafter, by e-mail plus meeting if desired.

This gives you 6 weeks (excluding the vacation period) to complete the assignment. Time will also be allowed within the last few practical sessions to work on the assignment and obtain guidance from your tutor.

1.2 Learning Outcomes Addressed by this Assignment

1. Demonstrate an appreciation of the architecture of a range of different microprocessors/microcontrollers and display a detailed understanding of a particular device.
2. Use a variety of software development tools to produce, test and debug software for small embedded systems in C and assembly language (partially)
3. Describe and apply best practice in software development for the use of a variety of system and user peripherals associated with a modern microcontroller.
4. Characterise and use specifications of typical transducers, actuators and network controllers to interface them to a microcontroller and develop device driver software to make such devices conveniently available for use by an application developer.

1.3 Academic Misconduct

The programaming task may be done with *one* partner; the report is is an *individual* piece of assessed work - do not collude with other class members. Please read the University Guidelines on Academic Misconduct.

1.4 Structure of this Document

The sections of this document are organised as follows. Section 2 describes Part A of the assignment; it provides the background to the problem, a specification of the control system in section 2.1. Notes on your implementation and a suggested algorithm for the game is presented in section 2.2. The stages of the assignment are outlined in section 2.3 and section 2.4 offers some technical hints. Section 2.5 gives the breakdown of marks. Section 3 describes Part B of the assignment - a short report.

2 Program: The Pong Game

The game area is a rectangular “court”. A small coloured ball moves inside the court, bouncing off the side walls and the top wall at the same angle as it struck. The ball goes out of play when it reaches the bottom wall of the court but above this is a small

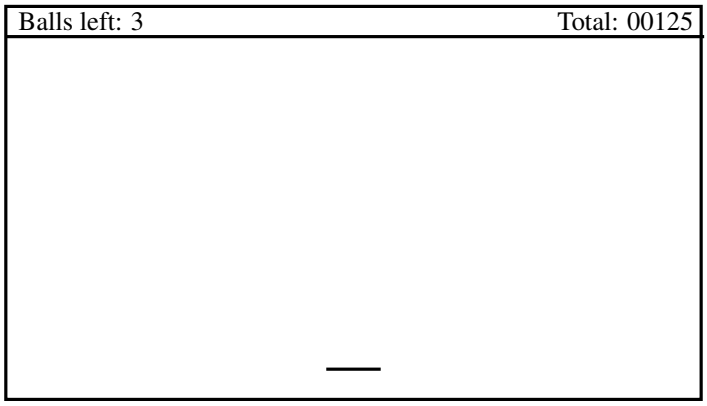
bat which can be moved from side to side by the player; the ball may be bounced off this back up into the court and remain in play.

The player earns points by keeping the ball in play as long as possible; the scoring rate increases as a ball remains in play.

2.1 Requirements

2.1.1 The playing area

The court is a rectangle within the display area of the LCD. Most of the display area of the LCD is devoted to the game display and is rendered in white. A top border displays the score and number of remaining balls, as indicated here:

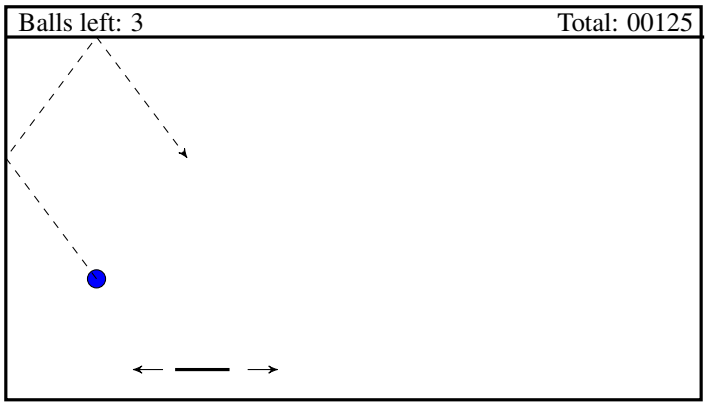


2.1.2 The bat

This is black, 40 pixels wide and 4 pixels high. Initially it is a random distance across the court and 10 pixels above the bottom. Its sideways movement is controlled by the joystick or by tilting the ARM board; it does not move vertically.

2.1.3 The ball

The ball is to be rendered as a filled blue circle 10 pixels in diameter. It moves within the court, bouncing off the side and top walls and the bat.



The bat movement must be controlled to catch and return the ball before it goes out of play.

The game is started by a press on the joystick CENTRE: a new ball starts moving down the court at a random speed and in a random direction.

If it strikes the side walls or bat, the ball bounces spinlessly – the angle of reflection equals the angle of incidence. (The easiest way of implementing this is to have one of the velocity components change sign while the other stays the same.)

If the ball is caught by the bat it bounces in the same fashion, up toward the top of the court. Otherwise it goes out of play at the bottom.

After a new ball the player earns a point each time the ball reaches the top of the court. After 5 such "returns" of this ball the scoring rate increases to 2 points balls for each return; after 10 returns, 3 points per return and so on. As long as the ball remains in play the scoring rate grows by a point every 5 successful returns.

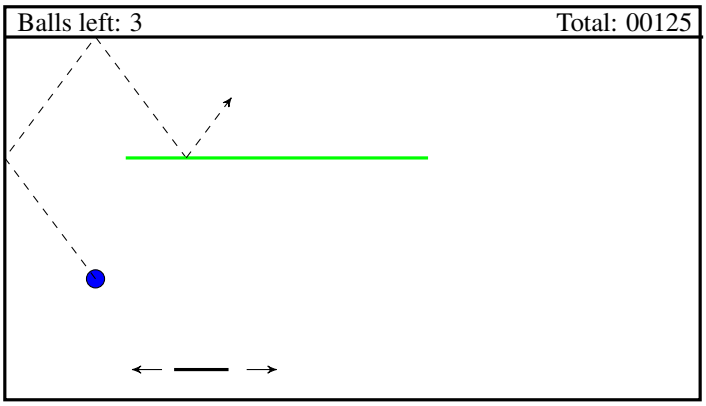
When a ball goes "out" the player can summon a new ball with another press of joystick CENTRE but the scoring rate is reset to 1. The score for "This ball" is reset to 0 but the "Total" score remains.

The game is over after 5 new balls. Joystick CENTRE will start a new game but *both* scores are reset to 0.

Successful software development to this stage will provide a basic pass of 50% for the run-time assessment.

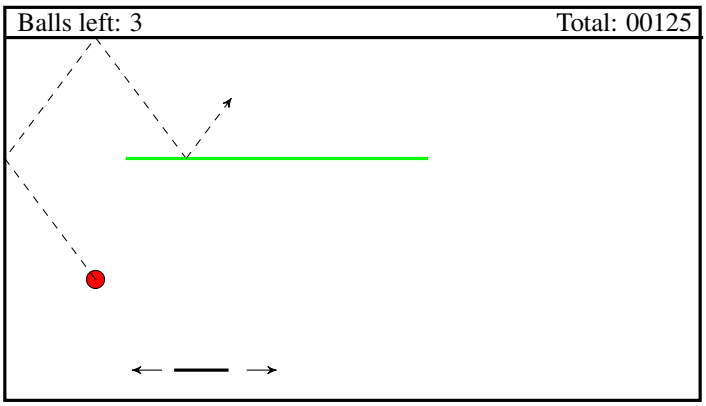
2.1.4 An Obstacle

A line of random length between 40 and 200 pixels (shown two pixels thick in green) is positioned somewhere in the top half of the playing area. The obstacle behaves like a wall: the ball bounces spinlessly off it. (But without earning points – only the top wall of the court counts for this.)



2.1.5 Magic Time

A “magic time” period occures at random intervals of between 5 and 10 seconds - i.e, between 5 and 10 seconds after the last “magic time” period, and lasts for a random duration of 2 to 10 seconds. During “magic time” the ball is displayed red and the scoring rate is double.



2.1.6 Ball speed

The speed of the ball may be controlled by adjusting the position of the potentiometer on the board. The control should adjust speed from “very slow” to “impossibly fast”.

2.1.7 Game Controls

- The bat is moved by moving the joystick left or right, or by tilting the board. Small changes in board angle should have no effect (small, random hand movements should be ignored).
- The game is started (and restarted) by pressing joystick CENTRE, which also delivers a new ball. A game consists of 5 new balls.

- The potentiometer position controls the speed of the ball.

2.2 Implementation

2.2.1 Constraints

Certain aspects of the C implementation must be adhered to and will be subject to assessment, namely:

Decomposition - your solution should show good decomposition (place different functionalities or responsibilities in different software components) - make good use of functions to encapsulate code in a sound engineering fashion. Your functions should be short (no longer than about half a page). The `main()` function should simply call other functions - to initialise the game and play the game, for example;

Coherence - similar tasks, related to each other, are handled by particular functions;

Low Coupling - modification of one subsystem should have little impact elsewhere;

Managed Complexity - your solution should be sufficient and complete but no more complex than justified;

Documentation - ensure that you document your code to the correct degree. This should include headings on each function - the purpose, algorithm (if appropriate), and comments before each section if this aids clarity. Statement comments will only rarely be necessary if your coding style is clear.

Global data - avoid global data declarations - make full use of parameter passing and return values for function calls. Some use of global data cannot avoided (shared data between the interrupt service function and the main code, for example). However, this must be reduced to the minimum and documented;

Identifiers - choose identifiers with care and be consistent in their naming;

Literal Values - avoid literal constants for such things as boundary limits, the position of the obstacle, etc, like this:

```
if ( ball.x > 76 )
   ...
```

use `#define` or `const` or `enum` to declare such values and make them very visible;

Use of a Timer Interrupt - a timer interrupt service **must** be used to manage the periodic occurrence and end of "magic time". A timer interrupt period of 1 second is appropriate. Pay particular attention to the safety of data access by both the interrupt service and the main-line code, and *do not* give the interrupt service routine too much to do. Setting certain status variables should suffice here.

2.2.2 Data Structures

You may implement any data structures you wish - for instance, for storing bat, ball states. Your selection of appropriate data structures *will* be assessed. For example, you might use *struct* records to hold related data - ball coordinates, velocity components, etc.

2.2.3 Algorithms and Data Structures

The following (incomplete) algorithm is suggested as a starting point for the basic game:

```
initialiseDevices();
remainingBalls = initialiseGame();

while ( remainingBalls > 0 ) {

   waitForStart();

   ballInPlay= 1;
   batPos = initialiseBatBall( ball );
   renderBall( ball );
   updateBall( ball );
   renderBat(batPos);

   while ( ballInPlay ) {
      renderBall(ball);
      updateBall( ball );

      batMovement = ReadBoardAngle();
      batPos = adjustBat ( batMovement );
      renderBat(batPos);
      ballInPlay = testBottom( ball );
      delay();
   }
}
```

Your `main` function should be no more complex than this for the basic game. Note the use of functions to encapsulate low-level detail.

For managing the ball, you might consider a record type as below.

Rendering the ball consists in drawing a filled circle of appropriate colour and radius at x, y.

Updating the ball might just be adding vx, vy (or a multiple?) to x, y and then adjusting vx or vy appropriately if a collision is detected. A collision with a verticle wall will mean `vx = -vx` and with a horizontal wall (obstacle, top of court, bat) will mean `vy = -vy`. Do not forget updating the score if a collision with top is detected – define a subroutine for this. Think how you will detect bounces.

```
typedef struct Ball {
   int x, y; //position coordinates
   int vx, vy; //velocity components
   int colour;
} ball_t;
```

```
ball_t ball;
```

To initialise the bat and the ball you might set the bat position initially to be half way across the court. The ball should initially be near the top of the court, a random amount across and random velocity components causing the ball to head down the court. The x-velocity should be less than the y-velocity. You will need to experiment with random values – see Technical Tips section below.

2.3 Work Plan

If you are struggling, it will be better to provide a partial solution which works properly than an unsuccessful attempt at a complete solution. Similarly, it will be better (and easier) to provide a partial solution that is properly documented, than a complete solution without any documentation, as the later stages of the assignment are more difficult than the early stages. You are recommended to complete the work in phases as follows:

1. Initialise the devices you require, buttons, LCD, accelerometer, but do this in an initialisation function called from main. Write a function to initialise the score, number of balls and the display. Test this by calling it from `main()`.
2. Design a data structure for the ball coordinates and velocity components. Should the colour be included? Write functions for initialising, rendering and updating the ball and bat. Test this part with a suitably simplified main loop.
3. Introduce a function to poll the joystick CENTRE and wait until it is pressed before starting a game or ball.
4. Write a function to move the bat in steps left or right in response to the joystick. Then extend this to move the bat in response to tilts detected by the accelerometer. Only the X value need be used for this. There should be no response to very small accelerometer readings: you will need to experiment to determine right cutoff value.
5. Write a function to update the score when a collision with the top of the court is detected.
6. Write a function to detect when the ball has gone out of play.

   Successful development up to this stage will give you a basic pass (50%) with the code run-time assessment.
7. Introduce the obstacle in the top half of the court. Extend the collision function to detect obstacle collisions.
8. Introduce a timer interrupt to manage "magic time". It is suggested that the interrupt service simply sets/clears flag variables to manage start and duration. The main-line code can access these (globals) to manage the consequent game logic and rendering. Update ball rendering and scoring functions as appropriate.
9. Implement any additional functions including the speed control.

2.4 Technical tips

- The standard C function `rand()` returns a random number in the range 0 to `RAND_MAX`, defined in `<stdlib.h>`. To produce a random integer `r` between 0 and `N-1` use

  ```
  r = rand() % N;
  ```

  Thus `a + rand()%(b-a)` gives a random integer between `a` and `b-1` (assuming `a < b`).

- Test each (small) addition to your program with care - do not write long sections of code before testing.

- Save your code using different file versions between major edits to allow recovery to an earlier version if disaster strikes!

2.5 Marking scheme

Marks will be awarded for *functionality* and for *program style and structure*:

Program Functionality, 35% is based entirely on how well the program works under test. The software test will be performed with you present to allow clarification of possible strange behaviour. Marks are broken down as follows:

- Basic Game (17%)
  - 2% - Game playing-area display, score and remaining balls
  - 1% - Bat rendering
  - 2% - Bat moves in response to accelerometer input
  - 2% - Ball rendering and movement
  - 2% - Initial ball moving down court with random direction, speed
  - 3% - Bounces off top, sides, bat working
  - 1% - Waits for joystick CENTRE before starting
  - 2% - Ball going out causes a wait for new game; remaining balls value is adjusted
  - 2% - Scoring
- Advanced Features (18%)
  - 4% - Obstacle displayed and collision of ball with it detected
  - 4% - Magic time comes and goes at required intervals
  - 4% - Ball and scoring during Magic time,
  - 4% - variable-speed bat movement
  - 2% - Game speed adjustable via the potentiometer

Partial marks may be awarded for attempted but incomplete (or poorly implemented) features. You will be asked questions concerning your solution and your code to confirm your ownership of the work.

Program Style and Structure 15% Based on the layout of the program, the use of comments and subroutines and the style of programming. Adopt a sound engineering approach in developing the code to aid the future maintenance of the software. Particular attention should be made to sound decomposition and use of functions. Marks will be awarded as follows (see notes in Section 2.2):

- 5% - Documentation, variable naming, layout and clarity
- 5% - Decomposition into functions
- 5% - Use of the timer interrupt to manage "magic time"

Marks will only be awarded here for a program that compiles cleanly and performs the basic game action. **Include a paper listing of the program.**

3 Report - Theoretical Aspects of ARM Programming

You are asked to address some theoretical aspects of your work, as follows.

Testing Strategy, 10% - What strategy would you adopt to test your software? Do not present test results - simply describe what testing procedure you would use at each stage in development to raise your confidence that the software would behave as specified.

Interrupt Safety, 10% - Although not a safety critical application, and malfunction of the game does not present any user safety concerns, the use of an interrupt introduces additional data integrity and concurrency difficulties. Describe briefly what additional problems are raised by using concurrency (interrupt code and main-line code) and what precautions you would take to ensure correct operation.

Timing, 10% - A feasible way to achieve a suitable time delay for the Pong game is ro use a *busy wait*. You have used this in lab exercises early in the module. Describe what this is, and contrast it with a solution based on *timer interrupts* – what are the relative advantages and disadvantages of each approach?

Input and Output on the ARM, 10% - Describe what memory-mapped input/output is, and how input and output are set up and performed on the ARM board. Describe in detail how LED1 is switched on/off and how a Joystic gesture is captured and reported to the the game software. What other things must the device driver software do?

Best Practice, 10% Discuss the principles of *best practice* that should be applied in general and in embedded programming in particular.

Your short report should be up to 10 pages in length - around 2 pages for each section. The resources listed on the module web page will be useful.