BE4M38AVS Application of Embedded Systems
Semestral Work – Ball game
Mikhail Ivanov

Introduction

The aim of the semestral work is providing of experience of programming embedded systems. The task was to create a game with usage of on-board peripherals - accelerometer, LCD, encoder, buttons, timers, PWM, EEPROM/RAM, UART/SPI communication etc. The game represents the classical ball in a maze game, however, electronic implementation let us to create different levels and add new rules – in the game you don't only need to bring the ball from start to finish, but you cannot touch the walls.

Methodology

The game mechanic is based on finite state machine. It let the game to switch between different states – logo, menu screen, campaign game and custom game. During a game, two separate timers provides an interface for LCD update via SPI (10 Hz was chosen as optimal for ball position update, higher frequencies would give an annoying blinking effect to the ball) and ball position updates based on communication with accelerometer (500 Hz). Based on the communication, we can define current acceleration. Sensitivity of 6g has been set in accelerometer control register via SPI communication. Ball updated position {x,y} is found in accordance will classical kinematic equations, but since the time step is set to 1, it has an simplified form:

$$x_{new} = x_{prev} + v_{x\_new}$$

$$y_{new} = y_{prev} + v_{y\_new}$$

Where the ball velocity v_{new} is defined as

$$v_{new} = v_{prev} + g$$

The ball position update algorithm also represents a friction by tolerating small velocities.

The program starts with logo screen that can be updated by menu screening with pushing of any button (BT1 or BT2). The menu position is set by encoder rotation, selection is possible due to buttons BT1 and BT2. The tree is shown on Fig.1.

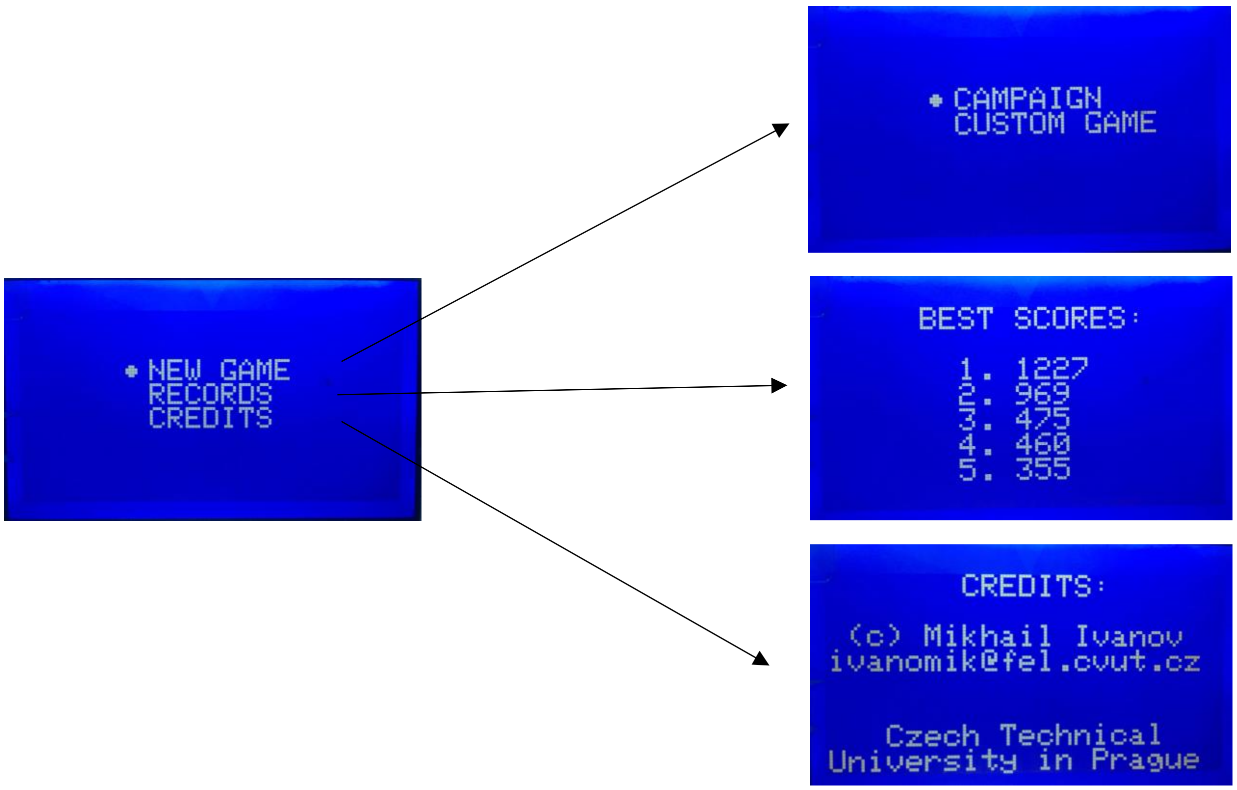

Fig.1 Menu tree

In the campaign mode the game has 6 different levels ranged in complexity order. The game levels are presented on fig.2. The aim is to reach the dotted area. Any wall collision will end the game.

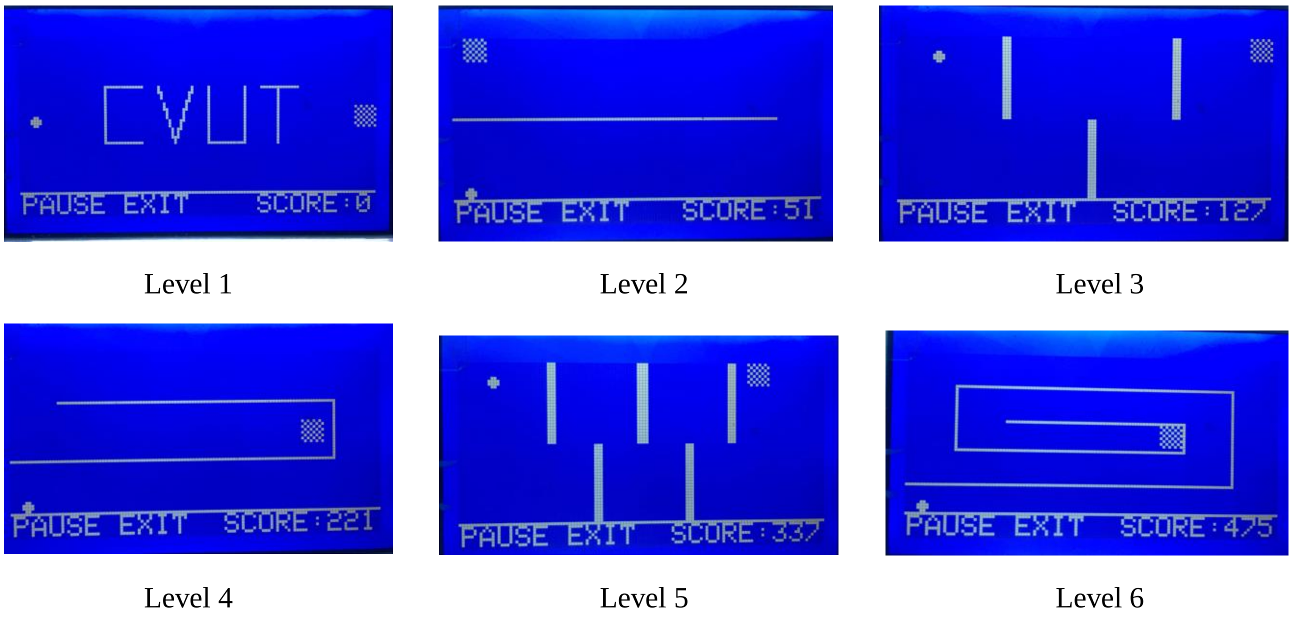

Fig.2. Game levels

Custom game is supposed to be uploaded via UART communication. The program sends appropriate instructions in terminal. The level map is sent by lines in blocks of 8x8 pixels size (represented as occupied or free), it’s also required to set ball initial position and finish area.

During the game, total score is calculated based on level complete time and current level (higher level will give higher score, numeration starts from 0). After each level complete it summed up with previous score. The score awarded for level completion is calculated as $3000 * \frac{level+1}{lcd\ update\ ticks} + 3$. Score records are non-volatile and stored in sector 11 of flash memory.

Conclusions

The ball game is a good example to get familiar with on-board peripherals. It let us to introduce all the skills we got from homework and create some fun debugging an own game. Despite of it’s first sigh easiness, the game uses different libraries for LCD, UART and accelerometer communication, flash memory library and standard C libraries. Usage of that libraries simplifies the task, however, the total number of code lines exceeds one thousand.

The game can be improved by logo animation, more sophisticated ball position update algorithm. It also possible to introduce more campaign levels and set different difficulties (higher sensitivity, “magnet walls” etc.).