Department of Information
Engineering and Computer Science

Embedded Systems for the Internet of Things

2020/2021

Labyrinth (marble game)

Matteo Busato

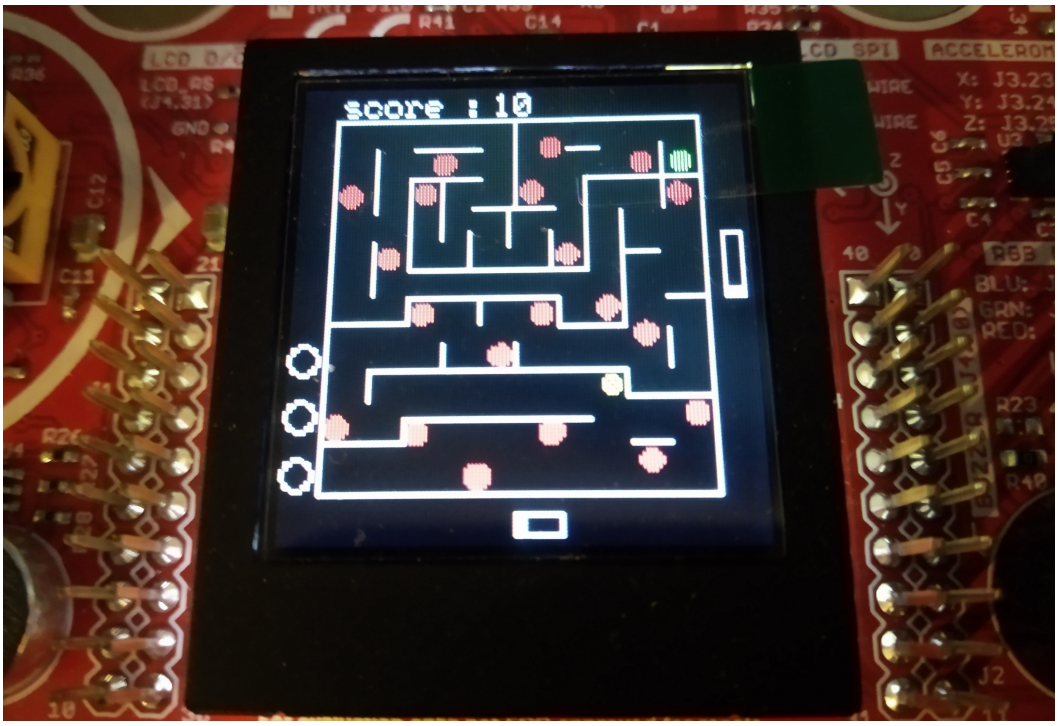
1. Problem Statement

Labyrinth is a game consisting of a box with a maze on top with holes, and a steel marble. The maze surface is suspended and rotates on two axes, each of which is controlled by a knob. The object of the game is to try to tilt the playfield using these knobs to guide the marble to the end of the maze, without letting it fall into any of the holes.

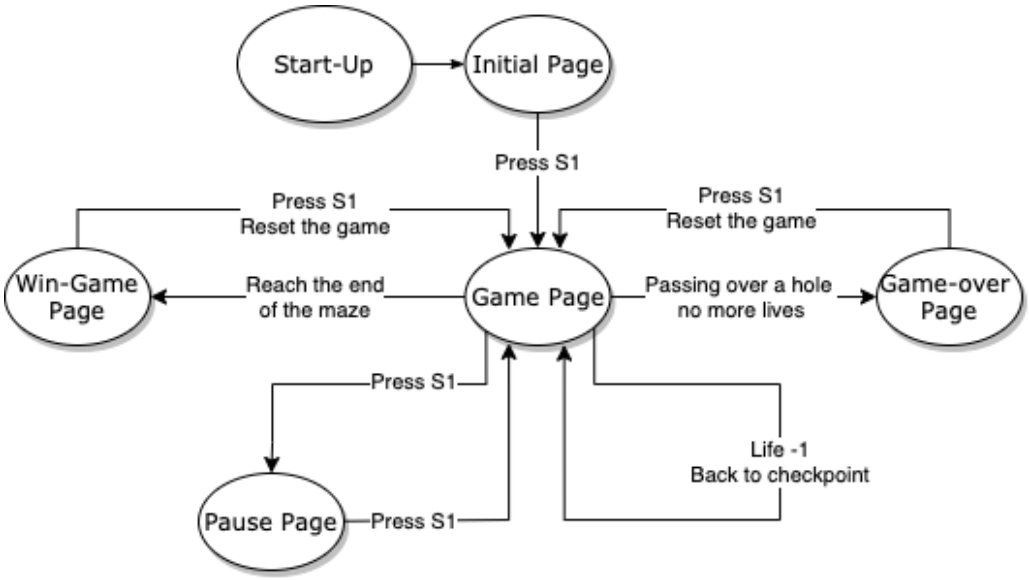
In this project, I digitize Labyrinth game, using the accelerometer sensor to control the inclination of the maze, resulting in marble's movements. I also implemented these other things:

- *Sound design occuring when the marble is positioned in some specific areas;*
- *A system of checkpoints used to restart the player to a specific point in the maze (so you don't have to restart from the initial point);*
- *A visual representation of the incoming data of the accelerometer sensor, so the player fit better with the control system;*
- *A three-life system so the player can have more possibilities to reach the end of the game.*

I made this project using Energia TI because of some problems on developing graphic things with driverlib, so I prefered to focus on hi-level programming to complete the projects in time.



2. Basic Working Scheme



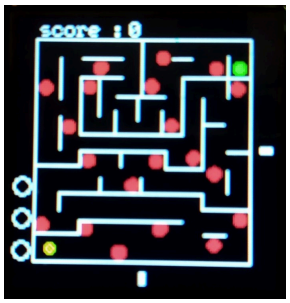
The game has some simple pages to fit the system workflow :

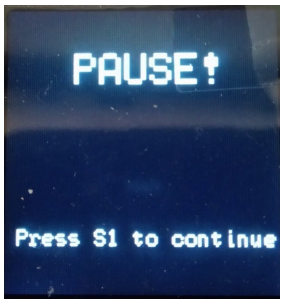
- **Initial page** : Show basic information about the game and it appears at the begin of the application;
- **Game page** : It is the core of the system, where the player can interact with the marble and move it.
- **Pause page** : Simple page to pause the game;
- **Game over page** : When the player fails three times, it is shown this page;
- **Win game page** : When the player reach the end of the labyrinth, it is whown this page.

If the player moves the marble over a red hole, life decreases and the marble is positioned on the last reached checkpoint.

Accelerometer data is collected on X and Y axis, mapped between 0 and 100 and compared to control the inclination type of the board (maze) :

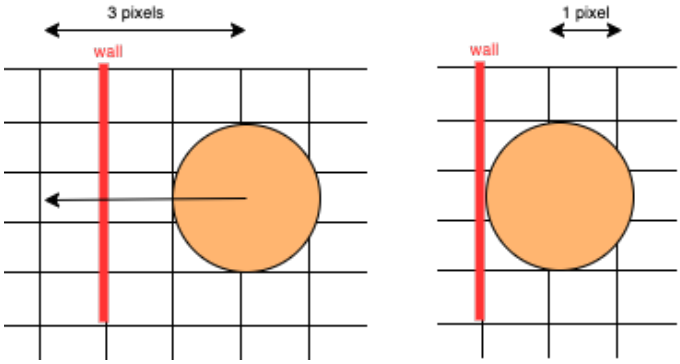
- **Left** : accelerometer **X** axis returns values from 0 to 49;
- **Right** : accelerometer **X** axis returns values from 50 to 100;
- **Up** : accelerometer **Y** axis returns values from 50 to 100;
- **Down** : accelerometer **Y** axis returns values from 0 to 49.





If the new values are different from the old ones, the accelerometer bars on the screen are refreshed and then the system checks the movements. According with the sensors data, the marble can move up to 3 pixels on X axis and up to 3 pixels on Y axis. Before the marble moves, the system needs to check if the movements interact with walls, to avoid overlaps.

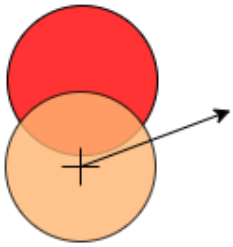
For example, if the marble wants to move by three pixels on X axis, but at the second pixel there's a wall, the resulting movement of the marble will consists of only one pixel.



After the system checks the movements, it draws the marble on screen following these steps:

- Delete the figure of the marble positioned on old coordinates;
- Restore the background around the marble;
- Re-draw the marble on the new position.

Then, the system checks if the marble is positioned over a red hole. If the center of the marble passes over a red hole, the player losts a life and the marble is positioned on the last checkpoint. So the marble can pass partially over an hole without penality.

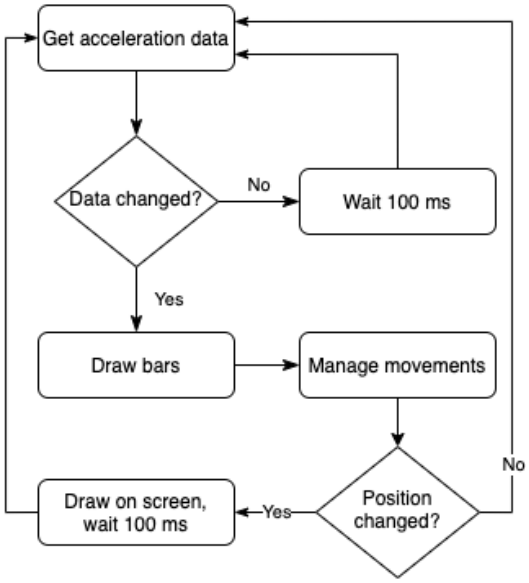


During the controls for walls, if the marble passes over a checkpoint, the scores level increases by 10 points and the system set it as checkpoint. There's 9 checkpoints in the labyrinth, 2 of them are not positioned along the main path of the labyrinth.

When the player reaches the end (visually displayed as a green hole), the scores level increases by 10 points and the win page is displayed.

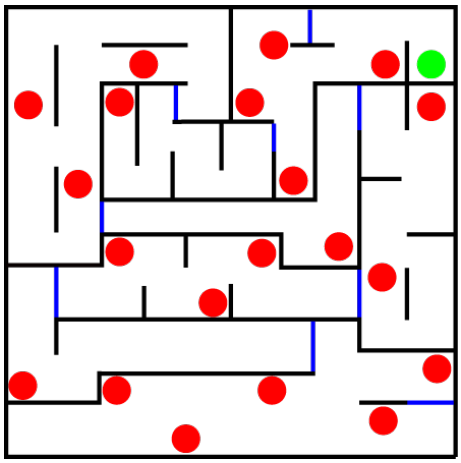
3. Software Architecture

While the system is in game page, cyclically gets the accelerometer data, if it changed w.r.t. old data then draws the bars and manage the movements. If the movements is possible, then draws the results on screen. The most important part of the system is the “manage movements” section that we will look at it soon.



3.1 How to draw the labyrinth

I “drawn” the labyrinth on Adobe Illustrator, then exported it in .png file and resized it in 106x106 pixels. I used “png2c” tool to transform the png file into a C array with exadecimal values. The result was not perfect because of the image resizing, so I manually checked the array resolving errors and managing the various blue lines (checkpoints) changing them to different values (like 10, 20, 30 and so on) to recognize different checkpoints during the gameplay.

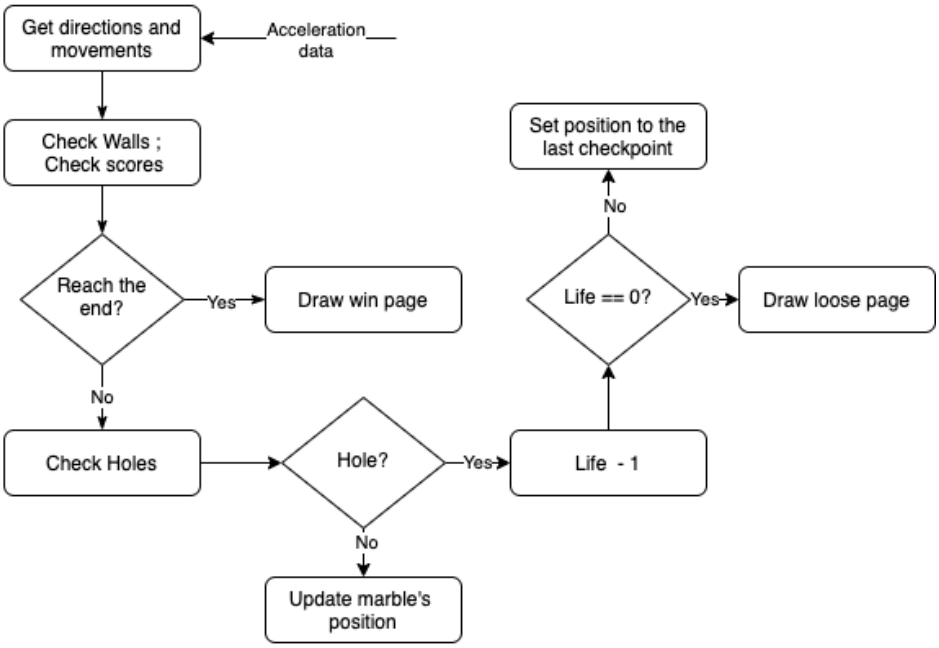


To draw the labyrinth on screen I just scan the array drawing point by point (excluding blue lines).

```
ffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff,
ffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 10, 10, 10, 10, 10, 10, 10, 10,
 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xff
 0xffff, 0xffff, 0xffff, 0xf800, 0xf800, 0xf800, 0xffff, 0xffff, 0xffff, 0xf
 0xffff, 0xffff, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xfb8e, 0xffff, 0xf
 0xffff, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xffff, 0xf
 0xffff, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xffff, 0xf
 0xffff, 0xffff, 0xf800, 0xf800, 0xf800, 0xf800, 0xf800, 0xffff, 0xffff, 0xf
 0xffff, 0xffff, 0xffff, 0xf800, 0xf800, 0xf800, 0xffff, 0xffff, 0xffff, 0xf
 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xf
```

The upper image shows part of the labyrinth array. You can see a hole into the red ellipse, a checkpoint into the blue rectangle and a wall into the green rectangle.

3.2 Managing movements



Data coming from acceleration sensor are mapped into 4 values :

- **dirX** : Direction on the X axis (0 = left , 1 = right);
- **dirY** : Direction on the Y axis (0 = up, 1 = down);
- **moveX** : Movement on the X axis between 0 and 3 (pixels);
- **moveY** : Movement on the Y axis between 0 and 3 (pixels).

The system checks walls and scores for x and y axes separately. Obviously the system does the check only if moveX > 0 or moveY > 0. This is useful to avoid the marble rolling over a wall, checking for every step starting from 1 to moveX (or moveY).

In the bottom pic you can see the code managing left movements. It returns the maximum possible left movements. We need to check all along the face of the marble and if we find a wall at i position, we return i-1 (the maximum possible left movements). It also controls the checkpoints for every pixels looked at.

```
uint8_t checkLeft(uint8_t &x) {
  int8_t i, j, X, Y;
  uint16_t pos;

  for (i = 1; i <= x; i++) {                    //control how much can I move to the left
    X = poswin[1].x - ANCHOR - i - 3;
    for (j = -2; j < 3; j++) {                  //control if I find walls upper and lower the marble's center.
      Y = poswin[1].y - ANCHOR + j;
      pos = *(imgPtr + ( IMAGE_H * Y) + X );
      if (pos == WALL) {
        return i - 1;           //return the possible movement.
      }
      if (checkScores(pos))     //if the player wins, I need to stop this procedure.
        return 99;              // 99 is only a number to check the winning situation from the caller scope.
    }
  }
  return x;                     //I can move without restrictions.
}
```

If the marble is moving, the system refresh its position on screen, then controls the holes.
The function `checkHoles()` takes the marble's coordinates and controls into the labyrinth array if there's a correspondence with a hole. If there's an overlapping with an hole, lifes decrease by 1 and the visual part of lifes are updated.

```
void checkHoles() {
  uint16_t pos;
  pos = *(imgPtr + ( IMAGE_H * (poswin[1].y - 10)) + poswin[1].x - 10 );

  if (pos == HOLE) {          //the marble's center position touches the hole
    life--;
    drawlife(life);
    if (life == 0) {         //you loose
      looseGame();
      looseSound();
    } else {                  //back to the last checkPoint
      holeSound();
      backToCheckout();
    }
  }
}
```

If life > 0 , the marble's position is updated to the last checkpoint passed. The system control what's the last checkpoint passed by scanning the checkpoints array (1 if you reach the i-th checkpoint, 0 if you do not).
The marble's coordinates are updated, the old marble are deleted and the new one are drawed on scren.

```
void backToCheckout() {
  uint8_t i = 0;
  while (checkpoints.visited[i] == 1 && i <= 9) {
    i++;
  }
  drawLabirynthArea(poswin[1].x, poswin[1].y);

  poswin[0].x = poswin[1].x;
  poswin[0].y = poswin[1].y;
  poswin[1].x = checkpoints.x[i];
  poswin[1].y = checkpoints.y[i];
  setBall(poswin);
}
```

3.3 Sensors

I used a simple structure to store X and Y values from the sensor.
I store both the actual data and the previous one to check if they are not equals to start other tasks. I mapped the data like explained on "Basic working scheme" section to understand the directions and the quantity of movements.I'm not using interrupt to start the collection of data, but polling methodologies instead.

```
typedef struct{
  uint8_t x;
  uint8_t y;
}Acc;

typedef Acc AccWin[2];
```

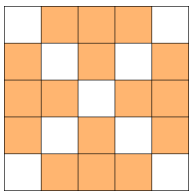
The button S1 is used to switch from a page to another. In this case I used an interrupt with a control system to avoid debounce problems.

```
void _initButtonS1() {
  pinMode(BUTTON_S1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_S1), go, LOW);
}
```

3.4 Screen tips

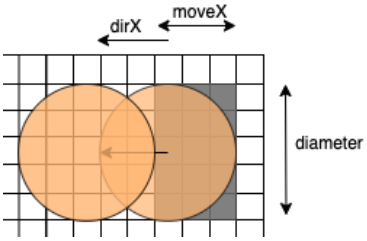
The marble figure is drawn on screen using predefined functions to draw circles. To delete the marble from screen I just use the same method changing the color to black (like the background).

```
void setBall(PosWin &poswin) {
  screen.circle(poswin[1].x, poswin[1].y, 1, yellowColour);
  screen.circle(poswin[1].x, poswin[1].y, 2, yellowColour);
}
```

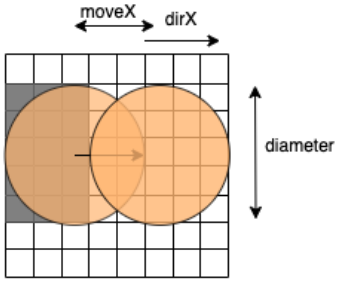


The problem with this method is that the marble can partially passes over a red hole and so the hole will also deleted with the marble. To restore the red holes I just use a system that collect the direction of the marble (dirX and dirY) and the movement (moveX and moveY) to know where to restore the background from the labyrinth's array. For example, if the marble is moving on left with 3 pixels, I have just to restore a 3x5 (5 is the diameter of the marble) rectangle on the right of the new positioned marble. The following part resumes the `restoreBackground()` method (the area to restore are colored in grey) :

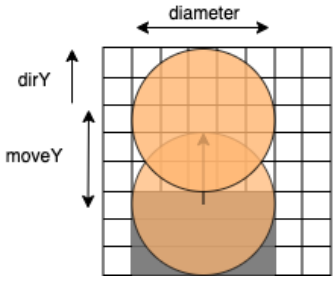
- **Marble moving on left, restoring right** :



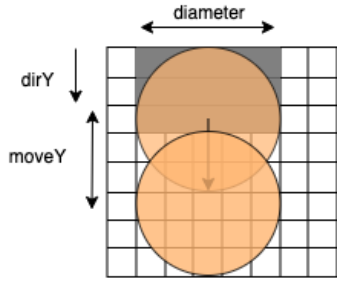
- **Marble moving on right, restoring left** :



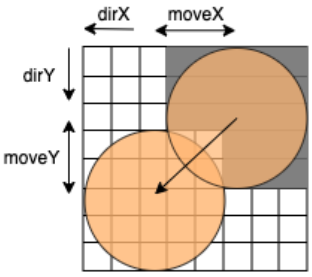
- **Marble moving up, restoring bottom** :



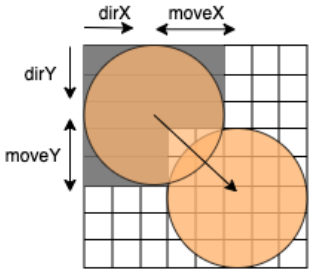
- **Marble moving down, restoring up** :



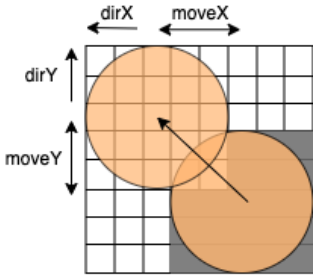
- **Marble moving left-down, restoring right-up :**



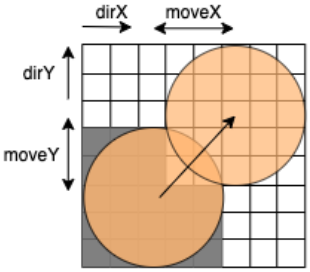
- **Marble moving right-down, restoring left-up:**



- **Marble moving left-up, restoring right-down:**



- **Marble moving right-up, restoring left-down:**



`drawLabirynthArea()` is another funcition used to restores a 5x5 rectangle from the labyrinth's array to the screen. This is useful when the marble passes over a hole and the user has to back to a checkpoint.

```
void drawLabirynthArea(uint8_t &x, uint8_t &y) {
  uint8_t i, j, limitX, limitY;
  uint16_t pos;

  i = x - 2 - ANCHOR;
  j = y - 2 - ANCHOR;
  limitX = i + 4;
  limitY = j + 4;

  for (volatile uint8_t yy = j; yy < limitY + 1; yy++) {
    for (volatile uint8_t xx = i; xx < limitX + 1; xx++) {
      pos = *(imgPtr + ( IMAGE_H * yy) + xx);
      if (pos == 0xffff)
        screen.point(xx + ANCHOR, yy + ANCHOR, blackColour);
      else if (pos == HOLE)
        screen.point(xx + ANCHOR, yy + ANCHOR, redColour);
    }
  }
}
```

4. Testing

Testing with Energia MT IDE was frustrating. There's no way to use a debugger. I just search on forums how to use some sort of debugging but nothing! In Energia there's the possibility to lauch the sketch with "mspdebug" tool but nothing work to me. So I tried to import the sketch on CCS to have some sort of debugging and it works. Basic debugging functionalities. I also tried to check the board comsumptions with EnergyTrace but it results with some graphical glitches and then CCS fails.

So I start looking for a simpler method : using `Serial.print()` to print data on serial monitor. This was just enought for my project even if the result was a sort of "try and error" technique.

5. Conclusions and Future Work

This project give me the chance to explore embedded programming. I am not very proud of my work as I have not used true embedded style programming, but I am very happy to have worked in this new environment. Using Energia MT allowed me to think only about the logic of the game, not so much about the board's architecture. So the system I built doesn't have a solid programming foundation for embedded systems (such as power saving and the use of interrupts).
I hope to be able to make improvements both at the logical level of the game and at the programming level.

Summarizing..

Game logic :

- Add a kind of physicality to the movements of the marble, as those implemented have little to do with its natural movement;
- Add interactions of the marble with the walls (like a bouncing ball effect);
- Add several levels with different grade of difficulty.

Programming level :

- Implement the project on CCS using their libraries;
- Implement interrupt on accelerometer sensor;
- Implement the system with low power mode.