Implementing Rolling Ball Maze Game using CC3200 with OLED and Accelerometer

EEC 172 Spring 2021 Final Project

Zaw Aung
University of California, Davis
EEC 172 - Embedded Systems
Davis, USA
jetaung@ucdavis.edu

***Abstract*—The CC3200 is used with the built in accelerometer and Adafruit OLED to implement a game that simulates a player rolling a ball through a maze.**

Keywords—CC3200, Accelerometer, OLED, AWS

I. Introduction

EEC 172 makes use of the CC3200 LaunchPad to demonstrate and experiment with the concept of embedded systems. With this CC3200, simple games can be implemented with the use of the built in accelerometer and OLED, as well as other components if desired. The WiFi module on the CC3200 allows the CC3200 to communicate with the AWS IoT service to fetch and store data.

For the EEC 172 final project, the CC3200 is to be implemented in some way in which it uses two-way communication with AWS.

II. Conceptualization

A. Inspiration

Upon the completion of Lab 3 in which we use the accelerometer to roll a ball, my first thought was how to use this concept to implement a game. So when I heard that we had to propose and design our own project for the final lab, I knew that I had to create a game.

I decided to expand upon the rolling ball in Lab 3 and implement a full game centered around that concept. Taking inspiration from Super Monkey Ball, a video game in which a monkey rolls a ball through obstacles and collects bananas for points, I decided that my game would have obstacles, danger zones, and coins to collect.

B. Game Mechanics

The player will use the controller to tilt the ball through the obstacles towards the goal. The ball will roll realistically in a way such that it will accelerate when tilted and decelerate when flat. When the player reaches the goal, the player will then be sent towards the next level. Players can collect coins that will increase the player's score. There will also be danger zones that when touched will send the player back to the beginning of the level and reduce the number of lives by one. Once the player has hit a danger zone three times, the game will end and a message with the score will be sent to the player.

III. Approach

The equipment used in this project are as follows:

- CC3200 LaunchPad
- Built-in Accelerometer
- Built-in WiFi module
- Adafruit OLED

A. Implementation

The accelerometer is used as the controller to tilt the ball through the maze. The OLED is used as the display, which will show the level as well as the number of lives, number of the current level, and the score. The WiFi module communicates with the AWS IoT service to store/fetch the scores and send messages to the player.

B. Pin Mux Configuration

- GPIO
  - GPIO07: Pin 62 Output for OLED OC
  - GPIO09: Pin 64 Output for OLED DC
  - GPIO25: Pin 15 Output for OLED Reset
- I2C for Accelerometer
  - SCL: Pin 1
  - SDA: Pin 2
- SPI
  - CLK: Pin 5 for OLED CLK
  - MOSI: Pin 7 for OLED MOSI
- UART
  - TX: Pin 55
  - RX: Pin 57

C. Creating the Ball and Rolling Physics

The ball that the player guides through each level is generated using the drawCircle function from the Adafruit_GFX.c. The CC3200 communicates with the accelerometer using the I2C protocol to detect the tilts of the controller. Using these tilts, the coordinates of the ball are updated and drawCircle is called again twice: first to erase the old coordinates of the ball by drawing a black circle where the old ball was and seconds to draw the new coordinates of the ball.

The rolling physics are generated using an equation that would increase exponentially as it is continuously tilted in a direction and would decrease exponentially as it is flat. The equation is as follows:

x = x + ((dx + rollx)/6)
y = y + ((dy + rolly)/6)
rollx = rollx/1.025 + dx/2.5
rolly = rolly/1.025 + dx/2.5

If a more unstable or more controlled ball is desired, the denominators in each of the equations can be changed accordingly.

D. Generating Walls, Coins, Danger Zones, and Goals

The functions within the Adafruit_GFX.c file are used to generate each of the structures within a level. The fillRect function is used to create white walls that the player has to guide the ball through. The fillCircle function is used to create the coins, danger zones, and goals throughout each of the levels. Each level layout is stored in a function called renderLevel(), in which the number of the current level determines which walls, danger zones, coins, goals, and player starting locations are chosen.

In order to detect whether the player has hit a wall, coin, etc. a function called checkBounds is called within every time the ball is updated. Within this function is code that stops the player if they have hit a wall, sends the player back to the start and take away a life if they have hit a danger zone, increase the score and erase the coin if they have hit a coin, and send the player to the next level if they have reached the goal.

This boundary checking requires many lines of code as you need to stop the ball from hitting every side of the wall and keep it on the appropriate side of the wall that they have hit. You must also account for every direction that the player might hit a danger zone, coin, or goal. The amount of code required depends on how many structures you decide to include within a level. Overall, the amount of lines I used to implement 4 levels is 691 lines. This number may seem daunting, but many of the lines are repetitive and easy to implement as the same formula is used for each level.

E. User Interface

A function called drawString was created that would take a string and loop the drawChar function to print the string inputted. This function is used to easily display text that would be shown in the user interface at the top of the game, such as “Level:”, “Lives:”, “HighScore:”, and “Score”. To display the number of lives, three white circles would appear next to the “Lives:” text at the top that would disappear one at a time when the player hit a danger zone.

The score starts at zero and is updated by one every time the player either touches a coin or reaches a goal. This is done by having an integer that would store the total score and a character array that would be updated with the sprintf function everytime the score is updated. After the score is updated, the score is printed over the previous score with a black background, efficiently erasing the previous score while also displaying the new score.

F. Storing/Fetching Data from AWS

The game communicates with the Amazon AWS service to store and fetch the high scores.

To fetch the high score, the HTTP GET function sends a get request to the AWS service. The function then parses through the HTTP response with the strstr function to find the first occurrence of the high score in the response and the sscanf function reads the high score into an integer that is used to display on the user interface as well as be compared to when updating the score.

To store the current score of the game, the HTTP POST function sends a post request to the AWS service with the current score and updated high score if the current score is greater than the previous high score. After the score is sent, the player will either receive a text message or email with their score of the game and the game’s overall high score.

IV. Complications

One feature that was going to be implemented was a timer that would count down and determine how much time the player has left before the game is over. However, when implementing this feature, the refresh rate of the game significantly slowed down as the time needed to be updated and checked every cycle that the ball is refreshed. This made the game essentially unplayable and thus the feature was removed.

V. What Would Be Done Differently

The code to detect if the player has hit a wall, coin, danger zone, and goal is very repetitive and extensive. To create a more efficient code that would require less lines of code, a function should be created that would be called every time if the player has hit these bounds to reduce the amount of code.

VI. Reproduction

A. Required Components

To properly replicate the project, the listed components below are required:

- CC3200 LaunchPad with USB cable
- Adafruit 1.5” SSD1351 128x128 RGB OLED
- WiFi connection
- Amazon AWS account

B. *Wired Connections*

The wired connections on the CC3200 are as follows:

- Pin 7 to OLED MOSI
- Pin 5 to OLED CLK
- Pin 64 to OLED DC
- Pin 15 to OLED Reset
- Pin 62 to OLED OC
- VCC to OLED Voltage
- GND to OLED GND

C. *Setting Up AWS Communication*

AWS communication is required to store and fetch the high score data.

1. Set up an Amazon AWS account and go to the IoT Core service to create a thing, generate/download the required certificates and keys for said thing and activate it.
2. Create a policy with the actions "iot:GetThingShadow" and "iot:UpdateThingShadow". Update the Amazon Resource number and select the *Allow* effect and click create.
3. Attach the policy to the created certificate. This allows for GET and POST commands via REST from the CC3200.
4. Convert the certificates from .pem to .der format using openSSL and use UniFlash to flash the key and certificates to the CC3200.
5. Create an SNS topic and create a subscription to the topic with a phone number or email.
6. Create a rule to forward a message to your SNS topic when the score is sent to the shadow device by the REST API.

D. *Update Code*

Update the Wifi access point information in common.h to have an internet connection in order to communicate with AWS.

In main.c, you must update the SERVER_NAME, POSTHEADER, GETHEADER, and HOSTEADER appropriately with the correct information of your thing as well as the DATE, MONTH, YEAR, HOUR, MINUTE, and SECOND with the current time in order to properly communicate with the AWS. The date and time only need to be within the last month or so.

VII. CONCLUSION

This project expands on Lab 3 where a ball is rolled using the accelerometer to create a fully fledged maze game with realistic physics of a rolling ball. This implementation was possible with the use of the CC3200 LaunchPad with the built in accelerometer to control the ball and Adafruit OLED to display the game. Although some altercations with the frame rate occurred, such as the implementation of a countdown timer, the game was created and works in a perfect manner.