Final Project Proposal: Build Your Own Labyrinth

University of California, Davis
22 May 2024

Jason Feng and Andrew Hoang
Professor Soheil Ghiasi
UC Davis College of Engineering
Department of Electrical and Computer Engineering
EEC 172: Embedded Systems, CRN 39462

I. Description

Build Your Own Labyrinth will be an isometric maze game where the player controls a marble through an obstacle course comprised of tiles of various heights. The game has two inputs for controls: the AT&T Universal Remote is used to navigate menus, while the LaunchPad's integrated accelerometer is used to control the marble. SPI will be used to display data on the Adafruit OLED, and I2C will be used to receive data from the accelerometer. AWS IoT is used to store level data, along with leaderboards for high-scoring users. There will additionally be a static frontend webpage hosted in GitHub Pages which will contain information about the game, such as instructions and a video demo, along with a level editor to submit custom levels. Finally, a Python backend will be used to connect the frontend's level editor to the AWS shadow.

The IR remote will be used to access the menus. GPIO interrupts triggered by the IR receiver circuit along with the Systick module will be used to parse IR input, and map each 32-bit sequence to a value. Because the arrow keys and OK are not read by the IR receiver, the number pad keys will be used instead, with `2` mapping to up, `4` to left, `6` to right, `8` to down, and `5` to OK. There will additionally be an option to change username, which will utilize the remote texting program from Lab 3 to enter text into the program, overriding the navigation functionality. During typing, `LAST` will be used as a backspace character, removing the previous character typed, and `ENTER` will be used to save the typed username. In both cases, `1` will be used to navigate back one page. `1` is chosen because it is the only number key with no letters mapped to it, and thus does not affect the username.

By default, there will be some sample levels. Levels will be encoded as a sequence of letters and numbers, with each character representing a single tile in the menu. Level data will be stored in AWS, with GET requests to retrieve level information to parse into a level. Each level will additionally have a leaderboard, which stores the usernames of the top 3 high scorers, along with their scores. This leaderboard will be updated when a user finishes a level with a higher score than leaderboard users.

Gameplay will consist of multiple levels, where the player must navigate a maze from beginning to end as fast as possible. Score will be based on time to reach the exit, starting at a high value and decreasing linearly as time increases, with a minimum score of 0. Accelerometer tilt will control the acceleration of the marble, which is used to calculate velocity. The marble will bounce off of walls and roll down slopes, obeying physics, until it reaches the goal.

The frontend integrates a level editor into the project webpage. The frontend will be written with Vue.js and Bootstrap for a clean and interactive webpage. It will be able to preview sample and user levels using `GET` requests to an intermediate backend, along with having a video demo and game instructions. Most importantly, the integrated level editor will be easy-to-use and interactive. It is still undecided what framework will be used to program the level editor. This editor will be able to update user levels using `POST` to the intermediate backend, but not directly to the LaunchPad's shadow.

The intermediate backend will be written in Python's Flask framework, and will accept `GET` and `POST` from the frontend, and makes `GET` and `POST` requests to the LaunchPad's AWS IoT shadow. This intermediate backend is written for security purposes- frontend users may be able to use inspect element to change the contents of `POST` requests. As a remedy, this backend acts as an intermediate step between the frontend and the device shadow, and will only `POST` to the LaunchPad if requests have valid data. The frontend will therefore not have the certificates required to `POST` and `GET` to the LaunchPad, and instead make requests purely through the backend, which does have credentials. This backend can be hosted either locally or through an AWS service, such as Lightsail or EC2.

II. Design

The LaunchPad application begins on the main menu, which has three options: change username, sample levels, or user levels. Options can be selected by using remote input to press one of three buttons, with the current selection being outlined in a different color. Upon calling the function to enter the main menu, the OLED draws the menu's graphics.

The change username function will change what name will be displayed on the win screen, along with the leaderboard. This function will utilize remote texting from Lab 3. This function begins by drawing graphics for username selection, and then enters a loop to scan and parse for user inputs until `1` is pressed. Numbers `0` and `2` through `9` will allow the user to type in an uppercase username with a maximum length of `10`, while `BACK` and `MUTE` will be used to delete one previous character and save the buffered username respectively. When `1` is detected, the function returns immediately to the main menu, which redraws the graphics.

The sample level select and user level select menus will be very similar: there will be a menu with some amount of buttons, one per level. At the beginning of the level select menu function, a `GET` request is made to the device shadow, obtaining updated leaderboards, which will then be used in a function to view leaderboards.

The button to view leaderboards uses the data previously read from the `GET` request and displays the top three users per level. Pressing `4` and `6` will go back and forward one level respectively, and display the data for that level. Pressing `1` exits to the previous menu: either sample or user level select, depending on which function called the leaderboard function.

If the user decides to enter a level instead of viewing leaderboards, the level's data string is parsed into a 128 by 128 grid of pixels: the entire stage. This helps with redrawing the marble, as only some specific pixels should be redrawn every frame, instead of the entire stage. Next, the program enters a gameplay loop which constantly reads accelerometer data, handles collisions, and detects if the marble falls off the map or touches the goal. If the marble falls off the map, its position is reset to the level's starting position. As usual, pressing `1` returns to the previous menu.

When the marble reaches the goal, a victory screen is shown. A `GET` is sent to the shadow to retrieve the most up-to-date leaderboards. Next, the user's current score is compared to the three scores in the leaderboards, and the top three scores are selected. The updated leaderboard is displayed to the user, and a `POST` request is sent to the shadow with the new leaderboard, updating the global leaderboard for all users to see. Pressing any key will return to the corresponding level select.

The central link of Build Your Own Labyrinth is the AWS IoT Thing Shadow. The LaunchPad and the frontend store data in JSON format in this shadow, and use `GET` and `POST` requests to access or modify it. Each level's information is stored as a JSON object with string `grid` for encoded level data, and object `leaderboard` containing three strings `user1`, `user2`, and `user3` for a leaderboard per level. User-created custom levels also have integer `timestamp` for the time the level was created: the earliest level is evicted when a new level is added.

The LaunchPad retrieves leaderboard information and level codes using `GET` requests to the shadow, and `POST`s updated leaderboard information to the shadow. It is controlled by the IR remote for menu navigation through GPIO interrupts, and the integrated accelerometer for gameplay using I2C. It outputs to the Adafruit OLED using SPI to display menus and levels.

The frontend indirectly interfaces to the shadow through an intermediate backend, which limits frontend requests to modifying only a subset of data. This intermediate backend makes `POST`s with new level data on behalf of the frontend, validating that level data is valid, along with handling which level this new level should replace, using timestamps. `GET` theoretically does not require this filter, but the frontend lacks the certificates to make `GET` requests, so both requests are handled through the backend.

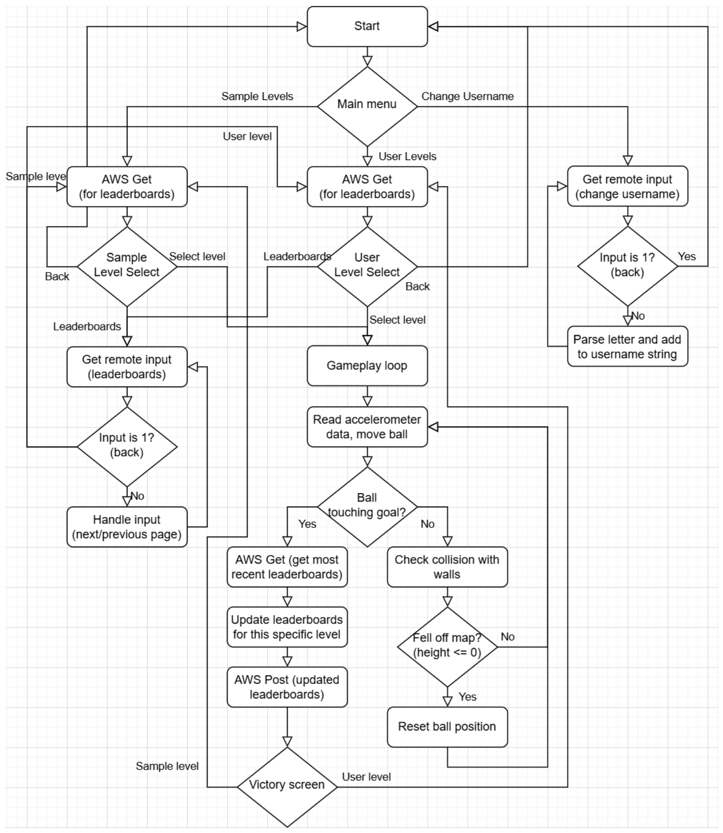
Start
Main menu
Sample Levels
Change Username
User level
User Levels
Sample level
AWS Get (for leaderboards)
AWS Get (for leaderboards)
Get remote input (change username)
Sample Level Select
Select level
Leaderboards
User Level Select
Back
Back
Input is 1? (back)
Yes
No
Select level
Leaderboards
Parse letter and add to username string
Get remote input (leaderboards)
Gameplay loop
Read accelerometer data, move ball
Input is 1? (back)
No
Handle input (next/previous page)
Ball touching goal?
Yes
No
AWS Get (get most recent leaderboards)
Check collision with walls
Update leaderboards for this specific level
Fell off map? (height <= 0)
No
AWS Post (updated leaderboards)
Yes
Reset ball position
Sample level
Victory screen
User level

Fig. 1: Build Your Own Labyrinth state diagram for LaunchPad code.

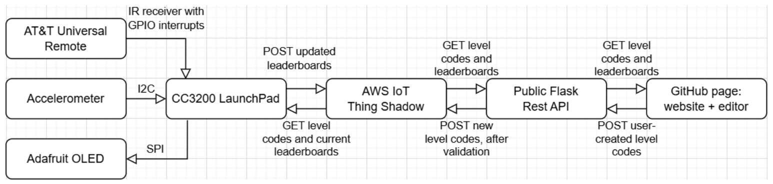
AT&T Universal Remote
IR receiver with GPIO interrupts
Accelerometer
I2C
CC3200 LaunchPad
Adafruit OLED
SPI
POST updated leaderboards
GET level codes and current leaderboards
AWS IoT Thing Shadow
GET level codes and leaderboards
POST new level codes, after validation
Public Flask Rest API
GET level codes and leaderboards
POST user-created level codes
GitHub page: website + editor

Fig. 2: Connected systems used in BYOL.

III. Implementation Goals

For this project we would minimally, like to be able to have the CC3200 be able to parse through user built maps on the OLED, allow users to interact with the marble using the CC3200 accelerometer, and a main menu that can be navigated using the IR remote. These minimal goals cover the bare minimum requirements of the final project. The CC3200 will be used as the core component, and will require AWS services to obtain the user made levels. The two hardware communication protocols I2C and SPI will be required to display and interact with the marble. Finally, the two sensing devices will be the IR remote and the on-board accelerometer both of which will allow for the user to interface with our software.

The target goals are to add 'sufficient' complexity to the project to set it apart from the bare minimum. In order to do this, our target goals are to have an AWS leaderboard which will save the top 3 performers of a level, an extension of the main menu which will allow users to enter/save a username, a frontend for users to submit their own levels, a python flask webserver which will act as a midpoint between frontend and AWS, and physics/slopes inclusions in the levels. These target goals will undoubtedly take a lot more effort/time and will require skills that are out of the scope of the class. Things such as the Python Flask webserver and physics systems will require knowledge from software engineering and frontend development.

The aim for the stretch goals for this project will essentially be to add game features and polish our product. Things like adding a visually appealing main menu screen, adding trampolines or powerups and including a coin system which will allow the user to buy marble skins. These are things that are not completely necessary when it comes to the functionality of the game/project but will be features that make playing more engaging. We hope that these stretch goals will go even further beyond complexities and push our project to be not only impressive but enjoyable.

IV. Bill of Materials

This final project is software oriented and will therefore not require any material that hasn't already been provided in class. Apart from the already required CC3200 Texas Instrument Microcontroller and AWS account, this project will require the AT&T S10-S3 remote, a Vishay TSOP 311xx/313xx/315xx Infrared receiver, a 100 ohm resistor, a 100 microfarad capacitor, and the Adafruit OLED display board. As stated before, all of these components have already been obtained from the EEC 172 lab therefore eliminating the need for additional funding.