BlockJump: DoodleJump (Arcade Edition) with CC3200 and AWS

Samuel Becerra Martinez [1]

Abstract

The purpose of this project is to create an arcade version of the widely popular iOS game Doodle Jump. BlockJump will allow you do play the game with similar mechanics to Doodle Jump. In addition, a user will be receive email regarding their score, and will have the ability to assign a name to a high score if applicable.

1. Introduction

There are a wide variety of games in the App Store today, but do you still remember the early classic games? With over 15 million downloads across all platforms, Doodle Jump pioneered gaming on your Apple device at an early stage. This game has a very simple, yet spectacular addicting, objective - which is to keep escalating the never-ending platforms without falling. This game bring great memories when I was a child, and with this motive I decide to make this my final project.

A variety of technologies have been used in order to have a promising product. We make use of the famous Texas Instruments CC3200 micro-controller and incorporate AWS to interact with the device using your phone in a 2 way manner. After the game ends, a user will receive an email with their score along with possibly other instructions to update the high scores table with their name if their score fits in the top five. A user will be able to email the games email and using AWS the high score table will be updated. This report will explain the method and development process of this application.

2. Equipment

2.1. Hardware

- Texas Instruments CC3200 LaunchPad (CC3200-LAUNCHXL):
This is used to perform the bulk of the operations and computation. In addition, we make use of the built in ccelerometer.
- Adafruit OLED Breakout Board - 16-bit Color 1.5" (product id: 1431) on small breadboards:
Used to display the game graphics along with other data like the high scores, and the setting to change the game's gravity.
- Jumper Wires
- BreadBoard

2.2. Amazon Web Services

- IoT Core:
Used to keep track of the physical device's state in the cloud (shadow)

- Simple Notification Service (SNS):
Used to keep track of our subscribers that the device should send emails to.

-Simple Email Service (SES):
Used to assign a domain (email) to the application. In this case we use @blockjump.net. SES also has a rule that will send and store all incoming email to an S3 bucket.

- S3 (Bucket):
Used to store all the receiving emails with the domain @blockjump.net

- Route53:
Used to obtain a domain name to work with SES (12 USD/annual)

-Lambda:
This services is used to have a lot of work done on the server side. We have two functions: 1.sendEmail which sends an email to a SNS topic (subscribers). This function will be triggered by a change in state in the IoT side. The email that is sent to the user will depend on whether the user achieved a high score or not. In order to determine this we look shadow state, specifically the 'hs' flag that was set to true or false after a user finishes a game. 2. UpdateState function will update the high scores that are displayed in the game with the name the user sends to the domain @blockjump.net. Thus, this function is triggered by a new incoming email from the S3 bucket.

[1]Department of Computer Science, University of California, Davis, CA, United States. Correspondence to: Samuel Becerra Martinez <subecerra@ucdavis.edu>.

3. Methodology

3.1. Code Overview

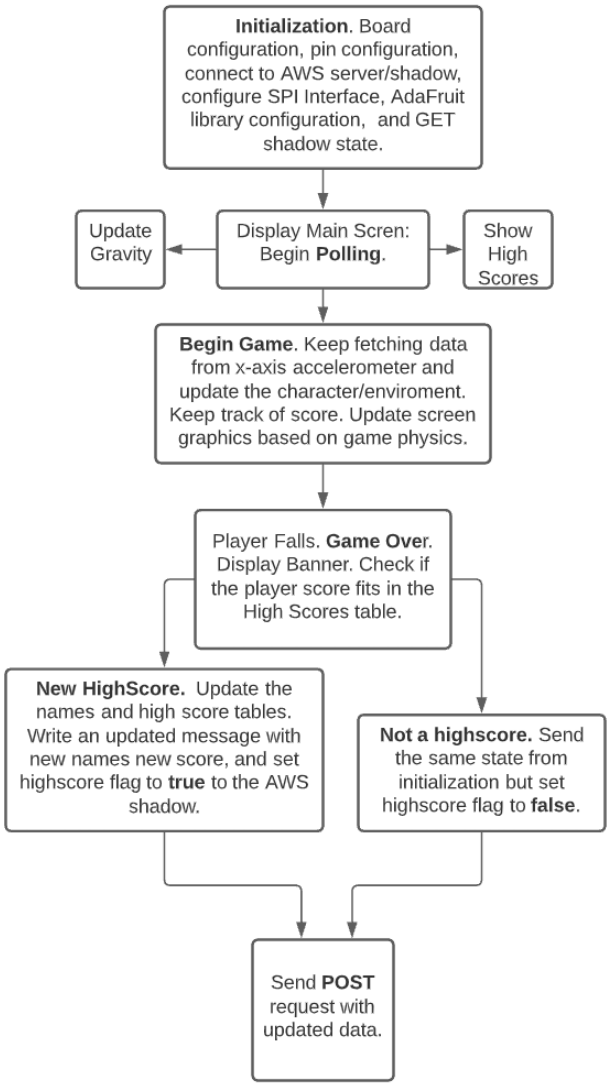

Figure 1. Code Logic Flowchart

Upon starting the application, we begin to initialize all hardware components like the board, pin configuration, UART, ADAfruit, OLED, and connect to the shadow using the RESTFUL API and get its current state. The game will boot up at the main screen where we have 3 options: play the game, show high scores, or change the gravity setting. During this time the CC3200 is polling for input from switches 2 and 3. To move down the menu we use switch 2 and to press 'ok' we push switch 3. If we choose the update gravity we again poll for the switch and change the gravity value accordingly. If we choose to display the high scores, from the state we got from the shadow, we parse the message and display the scores. After hitting enter, we go to the game with no option to go back to main menu unless we reboot the application. During the game we are constantly fetching the data from the x-axis accelerometer while also updating the graphics. The application will indicate that the game is over by displaying a banner and then checking if the current score is applicable to change the high scores. If the user achieved a high score, the high score tables will be updated by placing a dummy name and the user score. We then create a new payload to send back to the shadow. If the user does not score a high score, the initial state that was obtained during initialization stage is used as payload but we change a few variables. This message/payload is sent with the POST command.

3.2. AWS Overview

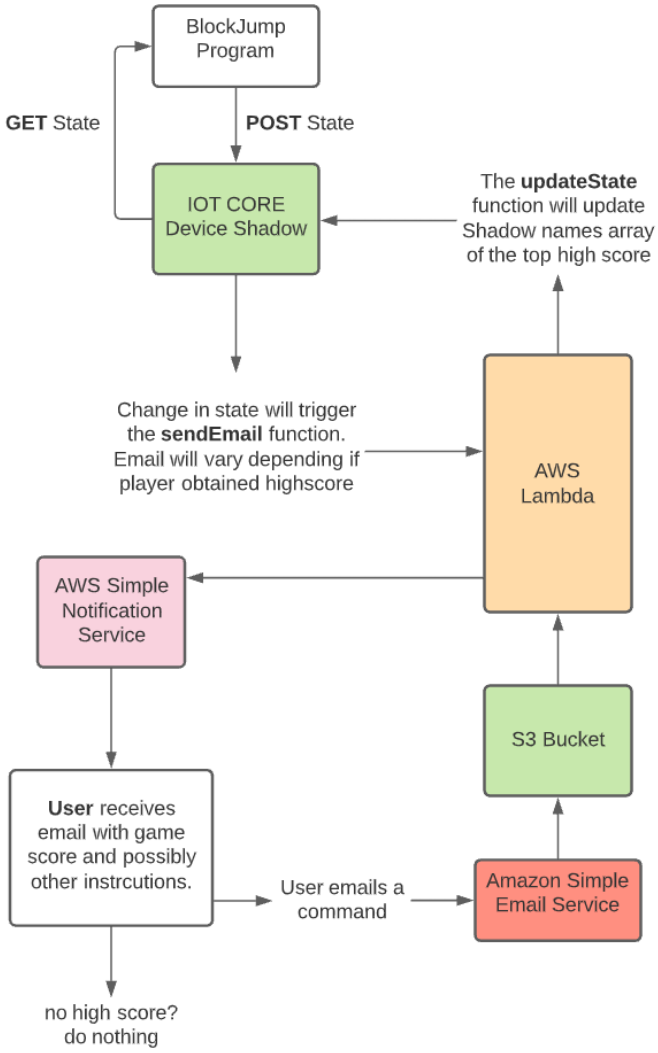

Figure 2. AWS Logic Flowchart

To keep track of the device state in the cloud we use IOT Core. A change in the shadow state will trigger the sendEmail function from Lambda. This function will email the user's score along along with additional instructions that depend on whether the user achieved a high score. The email is sent by using AWS Simple Notification Service. Using this service we have created a topic which contains all of our subcribers. If the user did not achieve a high score no further action is needed. Otherwise, the user will be sent an email regarding the option to assign a name to his/her high score. If the user does not want to change the name, the dummy name will remain constant. Otherwise the user can email scores@blockjump.net. This domain was obtained from AWS Route53 service. After the user sends her name with the following command: NAME 'insert score' 'insert name'. the Simple Email Service will direct this email to an S3 bucket for storage. When a new email is received in this bucket, it will trigger another lambda function that will change the high score table of the game with the name it just received from the email. This lambda function write back to the devices shadow and sets the highscore flag ,hs, to false so that the user does not receive another email because a change in the shadow will trigger the sendEmail function as previously discussed.

4. Conclusion and Future Work

Overall this application works as intended. The only change I would make is regarding the need to rotate the screen to play the game. This issue occurred because it wasn't until I finished coding the game that I realized that the ADAfruit library displayed the characters in a different axis. Although this is just a small inconvenience, it does take away from the user experience. If given more time, I would have been able to re-code the game mechanics. Overall, this application does serve its purpose and incorporates many techniques seen in embedded systems.

References

Various Users (2021, Late May-Early June). EEC 172 001-004 Piazza. Messages referenced from https://piazza.com/class/kjjppbl7ozoue

Lab 1-4 Manual. Handout. EEC 172-Embedded Networks. (Prof. Houman Homayoun)