Final Project: Remaking PONG

Ethan de Leon

*Abstract*— The main objective of this project was to implement 2-way AWS through SNS messaging in a fully embedded system with I2C and serial communication. The system implemented here is the game of PONG that was recreated in my own style and image.

# Introduction

The game I was trying to recreate is the classic game of PONG, but in my own style. I chose this kind of final project for two main reasons. The first reason was that I’ve coded videogames in the past as hobby and I felt that coding a videogame as my final project in the class would be fun and exciting. The second reason was that doing this as a final project felt like something that would naturally come next after completing the first four labs of this course. The main premise of the game is simple, a ball is bounced back and forth between the player and an object within a confined area. When the player fails to rally the ball back, the ball hits the wall behind the player and the game ends. I also added several different modes, levels and mechanics to the game. The player is able to communicate with the game via 2-way communication utilizing AWS to receive scores and messages and to select the game’s mode, level, and other options.

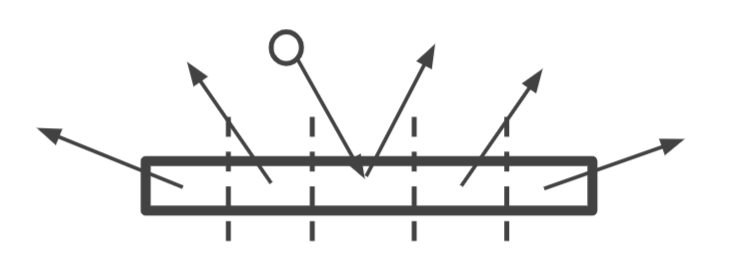
# Materials

The physical components that were used for this lab were the CC3200 Launchpad and the Adafruit OLED display, which communicate via serial communication. Software was developed using Code Composer Studios. AWS IoT, Pinpoint, and Lambda were used as a medium for 2-way communication between the CC3200 Launchpad and a mobile device. The mobile device that was used for this lab was an iPhone XR. The Launch has an accelerometer by which it communicates with 12C communication.

# Game Development and Methods

The project initially began as a copy of the source code for Lab 4. That project already had several elements that were needed for this final project, such as collision detection with the edges of the screen, a moveable object that was controlled by the accelerometer of the CC3200 Launchpad, connection to AWS, and a game over sequence. There were several steps taken to get the final results that I desired.

The first step was to modify this base code to create a game that was similar to that of PONG. AWS was temporarily removed at this stage of development so that focus would remain on making gameplay functional. The player-controlled object was changed from the ball to a rectangular paddle that lied near the bottom of the screen. The ball itself was no longer controlled by the player, but instead was set to move by itself. The ball’s speed would also increase over time until it reaches a maximum speed. When the ball reaches maximum speed, the ball would flash in rainbow colors. Ball collisions with objects like the walls and the player paddle were also implemented so that it would bounce off of them. The ball would change directions after colliding with the player depending on where the ball hit the paddle.

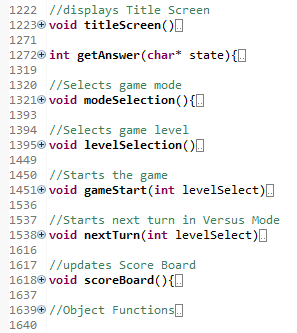


The only way to start the Game Over sequence was if the ball were to collide with the lower wall of the screen.

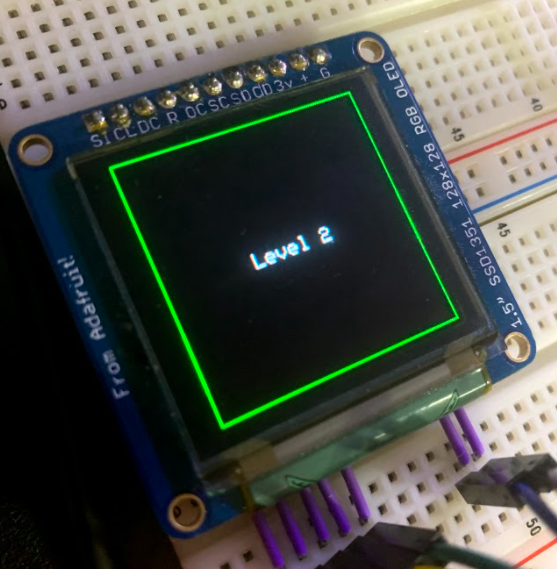
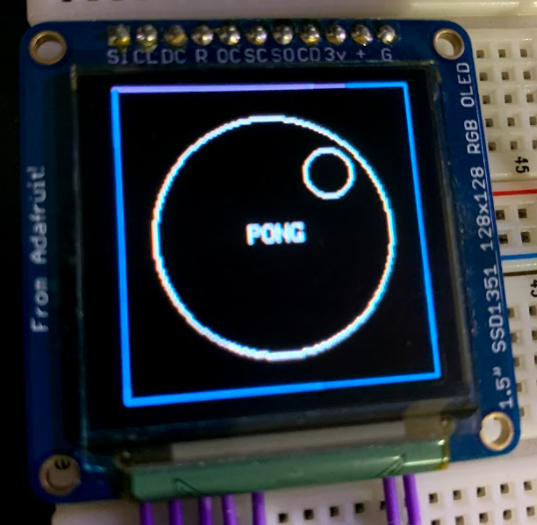
The next step was to implement different modes and levels that would change how the gameplay functions. This was done by creating functions that would be executed before the game starts to change different properties of the game, such as an object’s color, size, and speed. The first to be implemented were separate levels. Three levels of difficulty were implemented. Each level is able to manipulate the paddle size, the ball’s starting speed, change of speed, and maximum speed, and the ball’s color. As the level increases, the paddles size decreases while the ball’s starting speed, change of speed and maximum speed all increase. Also, at level 3, the ball begins the game already at maximum speed.

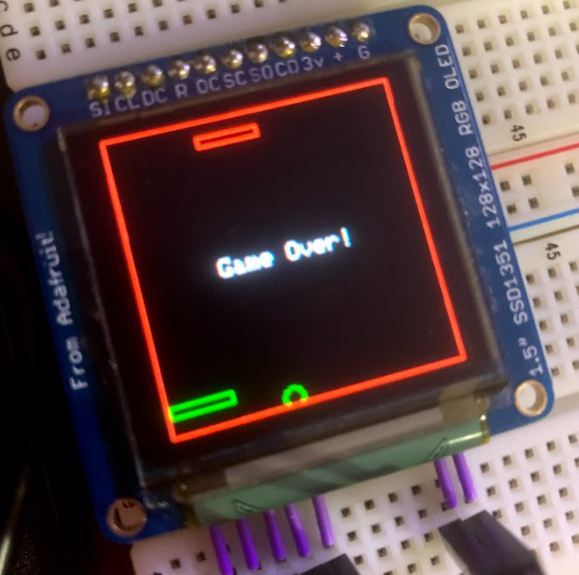
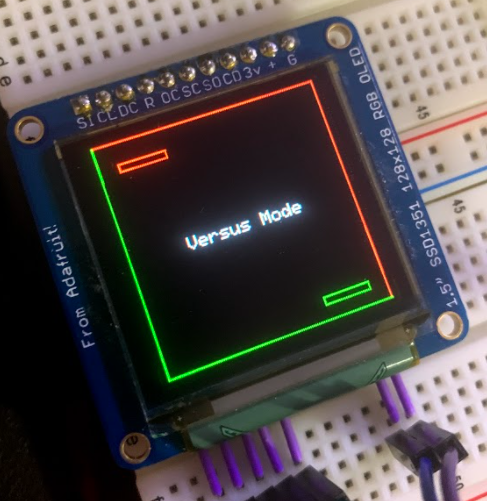
After the levels were implemented, different modes were implemented in a similar fashion. There were two separate modes created for the game: Endurance mode and Versus mode. In Endurance mode, the player tries to keep a rally with the ball for as long as they can. This was already implemented in the game, so no further action was needed for this mode. However, implementing Versus mode was a bit more challenging. In Versus mode, the player faces off an AI opponent, both trying to win by being the first to score 4 points. A turn system was implemented along with a score board to keep track of points. AI opponents were created to follow the ball at certain time intervals. Both modes were affected by level difficulties as well.

After both levels and modes were implemented in the game, the final part for creating the gameplay mechanics were to make the game keep track of the score and the time played. Keeping track of the rally score was simple to implement as a simple counter was implemented and would increase whenever the ball would collide with the player paddle. To keep track of time, the time library of functions had to be imported to the project. The time() function used for tracking time.



Above are the functions that were created to that make the game function. All of these functions are called in the main function to switch the game into desired states. A title screen along with a few other elements like game over text and transition screens that were added purely for aesthetics.



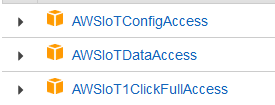


# AWS Implementation

After the game mechanics were finished and implemented, AWS functionally was restored to the project. SMS messaging was reused from Lab 4. Whenever a specific field in the AWS’s shadow updates, a message is sent to the player’s mobile device. To do this, a topic containing the mobile phone’s number was created and linked to an IoT rule.



In order for 2-way communication to be allowed between a mobile device and the CC3200 Launchpad, AWS Pinpoint and Lambda needed to be set up in our console. For Pinpoint, I had to set up a toll-free number that would allow for 2-way texting. For Lambda, I had to create a new SNS topic so that messages could be sent to this service. In order to change the launchpad’s shadow, the Lambda function’s configuration had to be changed. The following policies had to be attached to the project.

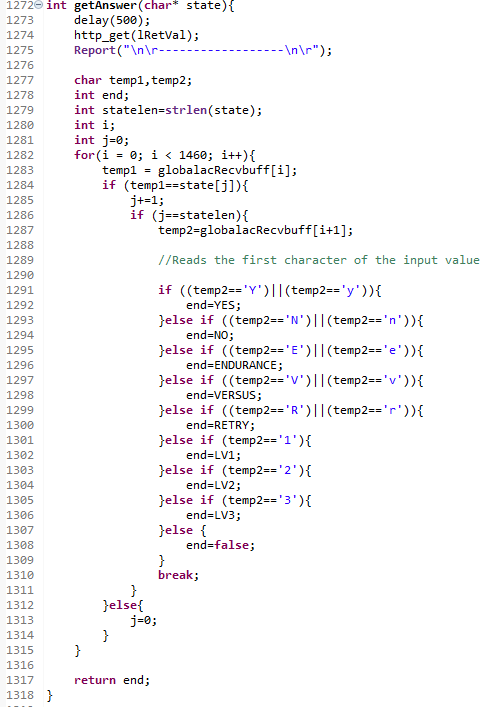


After the policies were attached, the following code had to be implemented in Lambda.



This code is triggered whenever a SNS payload is received. Whenever a text is sent from a mobile device to AWS Pinpoint using the toll-free number, a JSON message payload is sent to a desired SNS topic, which was the one connected to Lambda in this case. The code then parses through the JSON payload for the message sent from the mobile device and updates the launchpad shadow to store it as input for the Launchpad to read.

Initially, the game would only update the CC3200’s shadow and POST a message to a mobile device when the player receives a Game Over. This was changed to POST a message when player selects a mode, level, and receives a Game Over. The GET function was also coded into the project. Whenever the game needs input from the player, the launchpad polls the GET function to receive the AWS shadow JSON. In order to obtain the input for the game, the following function had to be initiated in the code.



This code searches for a specific JSON field and reads the first character of in the field’s value. If the character matches any of the desired input values, then the function returns a number associated with that input. If the character doesn’t match any of the appropriate inputs, then the function returns 0 or false. The program won’t continue until an appropriate input is read from the shadow’s JSON.

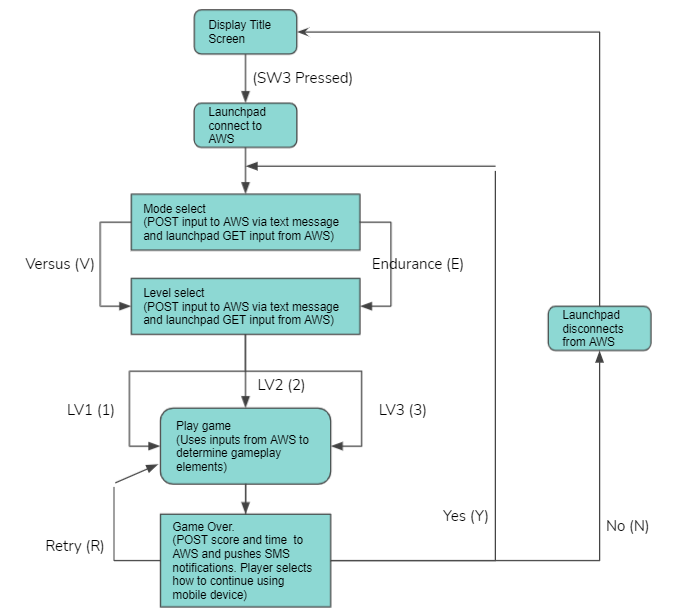
The project must also create a message to send to the mobile device once the game ends. The following code was implemented in the main function right after a game over is achieved. This function creates a message and then format the message into a JSON format.





The message that the code above creates holds information of the finished gameplay session. The message holds the game’s mode, level, time spent playing, maximum rally the player achieved, and the game over message, along prompts on how to continue the game. The message is then posted to AWS to be sent to the player’s mobile device via SMS.

# Algorithm

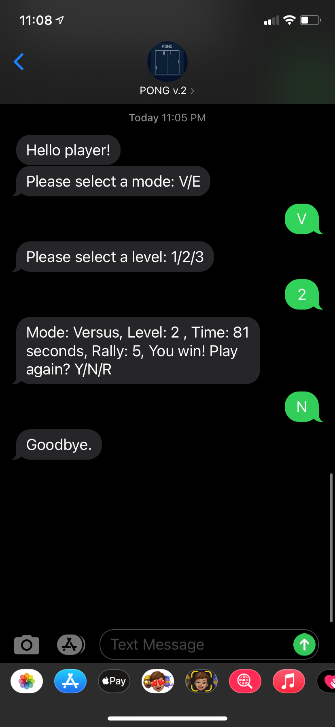


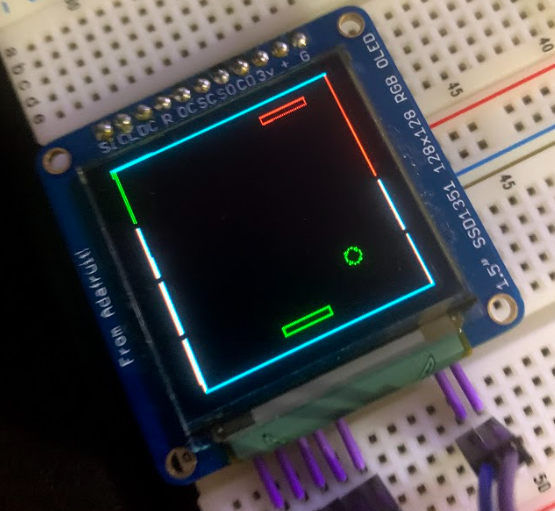
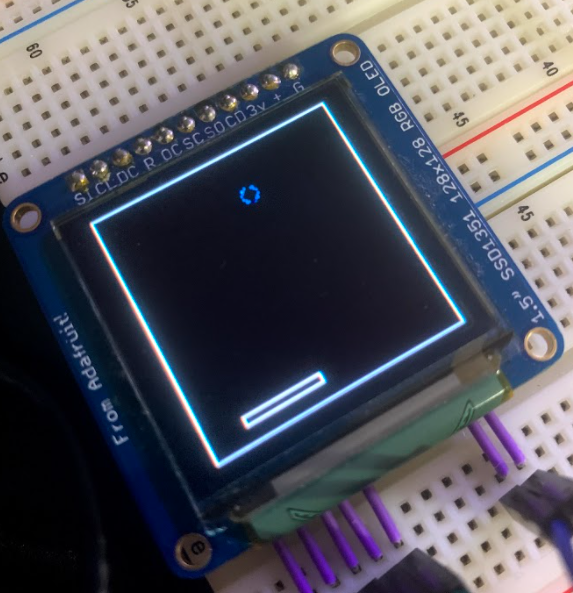
Above is a basic flowchart of the project’s algorithm. When the project begins executing its code, the player is first shown the game’s title screen. By pressing SW3, the launchpad connects to AWS and then the player is greeted and prompted to send input via text to select both the game mode and level. After the inputs are received, the game can be played. Once the game finishes, a message is created and sent to the mobile device through SMS. The player is then able to input how to continue with the game with their mobile device. The player has the choice of continuing by playing a new game with new settings, retrying the same game with the same settings, or disconnecting the launchpad from AWS and returning to the title screen.

# Challenges

There were some challenges that I faced along the way for this project. Displaying objects on the OLED were a bit challenging to utilize. I have worked on videogames before, but since I was not working with a dedicated game engine, the way of displaying objects on the screen is a bit different than what I was used to. However, I quickly adjusted. Another challenge I faced was when I reached the SMS limit in my original account, so for a while, I was unable to send messages to my phone. I sent a request to AWS to try to obtain more texts, but in the end, my request was denied. To fix this problem, I ended up having to create a new account.

# Results and Conclusions



Despite these setbacks, I’m pretty happy with the final results. The final version of the code ended up becoming quite long with a lot of different functions and states. The game runs pretty smoothly with no errors occurring. The frame rate of the game is very decent, the game doesn’t slow down all too much when multiple objects are displayed. The AI opponents in Versus mode could use some more tweaking, but at least both sides have the ability to win or lose. The project is able to send messages to the player via SMS texting and the player can send their inputs to the game. I believe this project successfully accomplishes this assignment’s objective of establishing 2-Way communication in an embedded system with I2C and Serial communication.

# References

1. <https://www.techiedelight.com/find-execution-time-c-program/>
2. <https://piazza.com/class/kjjppbl7ozoue?cid=343>
3. <https://canvas.ucdavis.edu/courses/568899/files/folder/Labs/Lab4?preview=12941295>
4. <https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-two-way.html>
5. <https://docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html>