EEC172 Final Project Proposal

Project Name: UCGuessr
Lab Section: A01
Partner 1: Harris Habib
Email 1: hshabib@ucdavis.edu
Partner 2: Yousif Nouman
Email 2: yalnomani@ucdavis.edu

Description

Our project is based on a game called GeoGuessr. The way it plays is a player is given a location of a place at UC Davis and then a map is loaded up on the OLED. The player then moves the reticle that's displayed on top of the map with a remote to where they think the image of the UC Davis location is and hits '5' on the remote to select it. The player will either score 0 or 10 points based on how close they are to its location. The player will do this until they've exhausted all images. Their final score is then displayed on the screen. This game was built upon the foundations of Labs 2 and 3.

Market Survey

a survey of existing similar products that compares and contrasts existing products with your prototype. You should provide at least 2 examples of other existing commercial products (provide make and model where applicable) that are related to your product, listing the advantages and disadvantages of each.

this should be more thorough than the project proposal and discuss particular features that you implemented in comparison with existing products

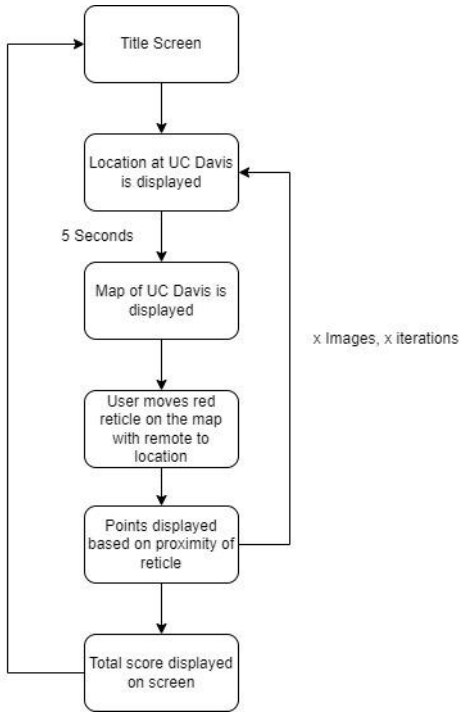
Multiple products are similar to the product that we made. The most similar product to ours is Geoguessr which is what our product was roughly based off of.

Geoguessr is a browser-based game that is simple in nature. It presents you with a pannable, zoomable, and movable image through 3D space. It does this by using the Google Maps API, which provides the application with street view coverage that it can use to make this possible. At all times, there is a geographical map displayed to the player so that they can make their guess. Once the guess is made, a point calculation is displayed to the user, and the original location is portrayed. It runs on a wide range of software, and hardware due to its accessibility from browsers. An internet connection is needed, and I/O devices such as a monitor, keyboard, and mouse are needed to interface with the game properly. Geoguessr uses a subscription-based model to provide its service

UCGuessr is roughly based on the same concept as Geoguessr. Our product uses the Texas Instruments CC3200 Microcontroller unit which is a Cortex M4-based board, an AdaFruit SSD1351 OLED screen, and a standard AT&T remote control. The key difference between our implementation of this version of Geoguessr is that it is based on the UC Davis campus, and has specific locations to guess for, unlike Geoguessr which is focused purely on the world map rather than a specific campus as in the case given here. The UI is mostly different too. While it does preserve some of the same design language and functionality as the original Geoguessr, due to the constraints of the hardware at hand a different approach to displaying information was undertaken. The OLED that is being used does not have ample space to display both a map and the location to guess, so we instead opted to display the location and shortly thereafter display the map on the same screen. This allows the player to view the location, and to guess it on the map in a similar fashion to the original. Another difference is the fact that the map is static in our implementation, and uses a cursor that is controlled by the remote control, rather than a draggable map. Similarly, the CC3200 cannot handle the Google Maps API, so we chose to implement a simple photo for each location rather than one that the player can move in. Lastly, our game does not require the internet to be played, but rather it just needs to be loaded onto the microcontroller.

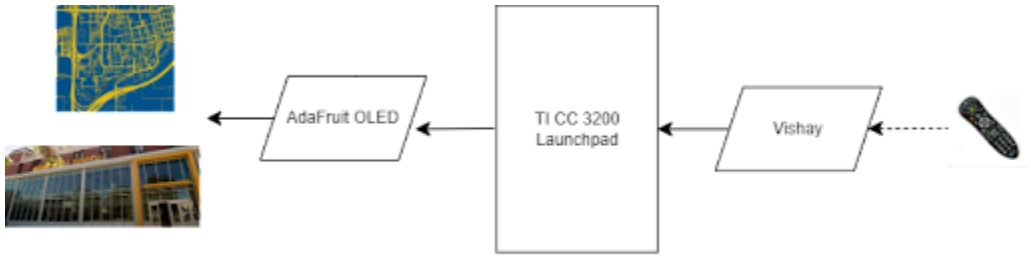
Design

Functional Specification



The game begins with a title screen that then transitions to displaying the location at UC Davis. After a brief 5-second internal, a map of the campus is presented. The user then interacts with the game by using the remote to move a red reticle. The buttons are:
2 = up, 4 = left, 5 = select, 6 = right, 8 = down. The goal is to align the reticle with our predefined location to accumulate points. The action is repeated x amount of times based on the amount of locations used. In this case, we used 2 images, so 2 iterations. The user will gain 10 points or 0 based on the proximity of the reticle. Upon completion, the final score is displayed on the OLED.

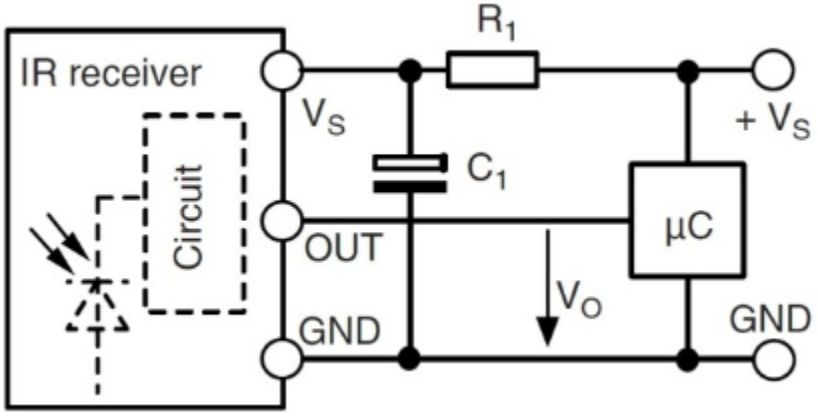
System Architecture



Overall idea: The remote sends input signals to the Vishay sensor. The Vishay sensor relays that information to the CC3200 as an electrical signal, which gets interpreted in software. Then, depending on what the input was, this determines what is being displayed to the OLED.

Implementation

Repository:
https://github.com/hhabib13/UCGuessr
Helpful software:
lcd-image-converter download | SourceForge.net



The AT&T remote control was provided to us. This remote control was programmed per the NEC encoding for the Zonda 1355 receiver.

'Vishay' is the IR sensor receiver unit that was also provided to us. The unit's schematic is provided above. V_s was connected to the 5V supply of our TI-CC3200 board. Ground was connected to a common ground with all other components, which was also on our board. And finally, the OUT in the schematic was connected to pin 63 on our board so that we could go ahead and listen to the signals from the remote. For listening to the remote's signals, we used a systick interrupt handler in our code. Inside the handler, the decoding of the raw IR signals (to binary) was done by interpreting the delays between each falling edge in the received NEC signal from the remote. If the delay was above a certain threshold, it would be interpreted as a 1, otherwise, it would be a 0. Using this information, we were able to interpret all of the encodings for each button pressed on the remote (0-9, LAST, ENTER, OK, etc.) into binary. This binary number would then be stored in a volatile variable, and decoded using the actual NEC encodings that were predetermined manually by us.

The AdaFruit OLED required 6 connections to interface with our microcontroller:

- MOSI
    - This is the most crucial pin as it transfers the actual data into the OLED
- CL
    - Clock
- DC
    - This allows the OLED to know when data is being written to it so that it can display it
- R
    - This is the pin required for resetting the display
- +, G
    - Power & Ground

The way we configured our pin multiplexing is that we had a pin assigned to the microcontroller board for each of those pins. The most crucial one being MOSI which was assigned to pin 7 by default by the OEM. The game logic was put into a while loop inside of the main function of our project.

This while loop would go on infinitely, as we give the user the option of replaying the game as many times as they like. Inside of the while loop, we continuously listen to input from the user as to know whether the game should start or not. In this case, the user must press a specific button (5) to start the game. The way this is done is by having a global variable that indicates whether there is an IR code that is received. Once that has been detected, a boolean is set to know that this title screen displayer portion of the code should no longer be entered, and instead, the first image is displayed for the round.

The image is displayed using a function called drawBitmap. drawBitmap is a function that takes in an array of 8-bit hex values (note: there are multiple ways to declare an 8-bit hex array in C,

we chose to go with a **char** type as it is interpreted as 8 bits across any compiler), a position given in x and y coordinates, and the width and height of the image. The width and height of the image are constant, as all images were fitted to be 128x128 so that they can fit on our OLED screen. This was done externally using third-party software. Additionally, using a software called LCD image converter, we were able to obtain the hexadecimal representation of our images in the RGB565 format. Inside of the function itself, the image is drawn pixel by pixel using the drawPixel function provided by the library. This is done by looping through each row of pixels of the hex array provided and drawing each pixel in that row. A key caveat for drawing the pixel correctly is that it is required to bitwise SHIFT the hex value that is in the array by 8 bits, and then do a bitwise OR operation to that shifted value with the value that is in the next index in our array.

Once that first image is drawn, there is a delay given so that the user can ponder the location, and a boolean, drawPhoto, is set to indicate that the location does not need to be drawn at this time. Once that delay is over, the map is drawn and a boolean is set in our code to indicate that the map is drawn, and does not need to be redrawn. Now, the user has the ability to move a red reticle that is displayed on the map in order to make a guess. The code knows not to proceed with anything until button 5 is pressed, which indicates that the player has made their guess. A score calculation ensues after the guess based on the location of the reticle, which is obtained by tracking the number of times the corresponding left, right, up, and down buttons were pressed by the player on the remote. Additionally, a counter is incremented to indicate that a round has gone by in our game, and drawPhoto gets reset to 1 to indicate the need to draw a location once again.

This logic continues until the counter hits the maximum number of rounds, which then takes us to the end-game logic. The end game logic simply resets the booleans to their pregame values, and displays the score promptly, after which we reach our start screen again.

Challenges

The first challenge that we faced was we were struggling to get the microSD card slot embedded into the AdaFruit OLED to be read by our CC3200. We followed the CC3200 launchpad datasheet and found out that certain pins (pin 6,7,8) all had 5 virtual pins assigned to them and could be selected through sysconfig. All were specifically assigned virtual pin 8 as the SD card reader. After making those connections we also needed to connect the MISO pin. We still couldn’t get a read on the card. We also used a TI project called SDhost (specifically the Fatfs version) which has documentation on how to read the microSD card from a different breakout board but we still couldn’t manage to get it working. Contacting TI’s forums proved to be tedious as the replies took a day on average and were not very constructive. A possible reason why the SD card was not working could be because the SDhost examples use a different communication protocol than that of the OLED, which uses SPI.

The 2nd challenge we faced was trying to flash our board. Since we got our photos the unconventional way into our game by converting them into BMP and then making them bit

arrays, we would run into the error of using up our RAM. What we did was we compressed our images multiple times and also used 3 of the 5 images that we were going to use. This allowed us to run the program in the debugger without having the issue of not enough memory being available in the microcontroller.

The 3rd challenge was flashing the board itself. We were able to flash and get the successful screen, but in return, the boards' MOSI pin would become dysfunctional. Therefore, we weren't able to check whether our game was loaded onto the board. We have no idea as to why this would happen as we tested this with two other boards and recorded the entire process. Checking with the logic analyzer showed LOW for the MOSI pin as well as the CLK being inconsistent with its behavior. All other pins that the OLED display used were working as intended.

Future Work

One feature we would like to add is adding a few more drawn graphics to our game. Such as a more aesthetically pleasing title screen and an instruction screen. Also, a timer would be nice to get the player to choose the location on the map under some stress.

We were also hoping to implement another board so that we could have one board displaying the image and the other displaying the map, so the player can look at both screens simultaneously while making their decision.

Later down the line, it would be cool to add other campuses within the UC system to the game and have the player select which campus they want to play on. It could also be fun to have randomized locations of those campuses and to have the person guess which campus it is out of a list of 4 options.

Adding AWS implementation into the final score would also make the game more intuitive and competitive through the implementation of a leaderboard. Due to time limitations, we were unable to implement the leaderboard but it would have been feasible to implement given a little more time.

Bill of Materials

Component Name	**Cost**	**URL**	**Notes**
CC3200-LAUNCHXL	~~$66~~	Digikey	Provided
Adafruit 1431 128x128 RGB OLED	~~$39.95~~	Adafruit	Provided
Remote Control/Vishay Sensor/Resistor/Capacitor	Free	AT&T/lab	Provided

Jumper Cables	$0.57		Harris Provided
Micro SD	12.99	BestBuy	Harris Provided, unused in project
Breadboard	Free		Provided