SodaShop Project Report

G185

Justin Kleen, Austin Koenigs, Paul Litherland, Liam O’Malley, Nic Witulski

5/6/2025

University of Nebraska-Lincoln

Department of Electrical and Computer Engineering

ECEN 494/495

SODA
SHOP

Abstract.

An automated soda machine with controllable flavor mixing is designed. The system takes a user selected flavor input via a touchscreen and dispenses the concocted flavored drink in a user placed cup. The system uses a Raspberry Pi, microcontroller, control relays, and pumps to fill the cup with carbonated water and flavored syrup. The end result is a soda machine that can create a drink combining up to four flavors at the push of a button.

Table of Contents

Introduction

Background and Motivation

Automated soda machines exist in many settings, especially fast food and food preparation places. However, there is no available automated soda machine at an affordable price for the average consumer. The motivation behind this project is to create a soda at the press of a button, including carbonation and soda flavoring.

Problem Statement

Cans of soda and other soda mixing machines do not allow for customer customization and control. SodaShop allows for the ultimate customizability of soft drink creation bolstered by seamless automation.

Objectives

SodaShop will integrate the customizability of modern soda machines found in restaurants with the ease of access of having soda from the comfort of your home. Additionally, SodaShop will utilize a touchscreen with a friendly user interface which features flavor customization of up to four flavors, while also adjusting the amount of syrup dispensed in order to provide further customizability. Finally, SodaShop aims to lessen the cost of buying CO2 canisters by utilizing club soda bottles.

Scope

SodaShop will focus on the pumping of carbonated water and syrup, designing a user interface for a touch screen and back end software to communicate between a RP2040 and a

Raspberry Pi 4. It will focus on designing a sleek enclosure for the machine and all the parts to interface with the store bought parts. This project will focus on designing a PCB motor controller with a RP2040 as the microcontroller. This project will not focus on developing relay boards, motors, pumps, and parts that would be far easier to buy off the shelf.

System Design

Overview

The Sodashop system is an automated drink machine that requires only a cup to be placed and a selection to be made. It then pumps carbonated water and flavored syrup into the cup based on the selection, for the user's enjoyment. It does this using a touchscreen display for the button, and a microcontroller to manage the amount of carbonated water and syrup that is pumped.

Hardware Components

The system consists of a Raspberry Pi with a 7” touchscreen, which uses USB serial communication to communicate with a custom RP2040 microcontroller board. The microcontroller board controls relays that activate peristaltic pumps for adding flavoring, as well as air pumps for adding carbonated water. The carbonated soda water is going to use an air pump to increase bottle pressure to pump water. The air system will have a pressure relief valve to prevent over-pressurizing of the bottle to prevent equipment failures.

When selecting components, it was chosen that the pumps and relays would run on 24 VDC to reduce the overall current draw without reducing power. Peristaltic pumps were chosen due to their positive displacement and precise fluid displacement based on the motor rotation, providing precise volumetric control via pump timing.

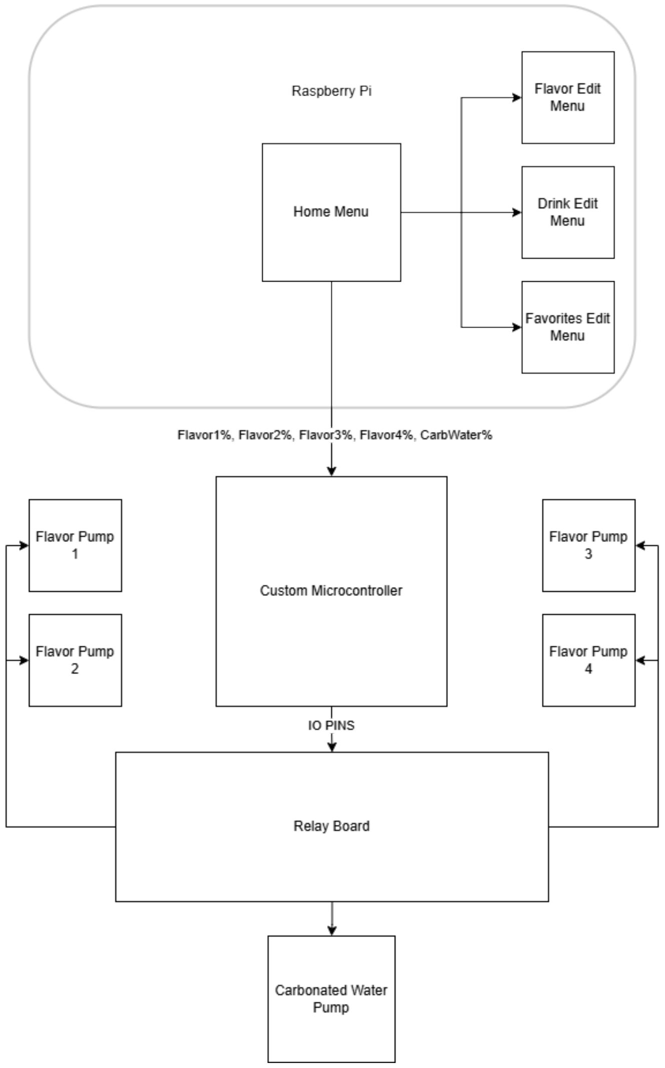
The relay's were chosen based on the 24V requirement with current draw of the pumps in mind. The selected relays are contained on relay-control PCBs with four relays per board. Each relay contact has screw terminals for simple connection and modification. The benefit of using a circuit board with relays is that they can share a common power and ground, with only the signal and output wires running to the actual relays. This helps to reduce the wiring and power supply outputs required.

A Raspberry Pi was chosen for the central computer for the power and functionality it provides for its cost. A Raspberry Pi 4 was chosen over a Raspberry Pi 5 as it still provided all necessary function at a lower price point, and its compatibility with the Raspberry Pi 7" touchscreen display. The Raspberry Pi 4 is also a great candidate for programming in Micropython, and supports a USB Serial interface for communication. Since the Raspberry Pi is being used only for the interface and serial communication, there is no risk of over-utilizing the processor.

An RP2040 microcontroller was chosen for the custom PCB as it integrates well with the Raspberry Pi and micro python environment. This also allowed the use of a Raspberry Pi Pico for development, since it uses the RP2040 microcontroller. This cuts down on the development time of the project, since the custom microcontroller circuit board isn't required to test the serial communication and GPIO outputs, and the pins used for the Pi Pico and the custom board can be the same.

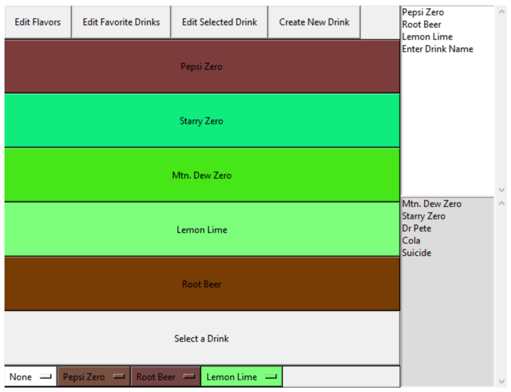
The power supply for the 24V system was chosen because it has a 6A maximum current draw, which is more than enough to run the four peristaltic pumps and the air pump simultaneously.

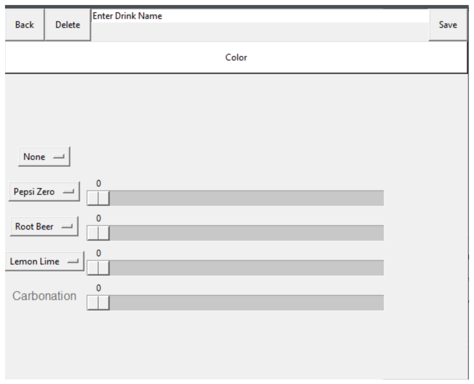
Software Components



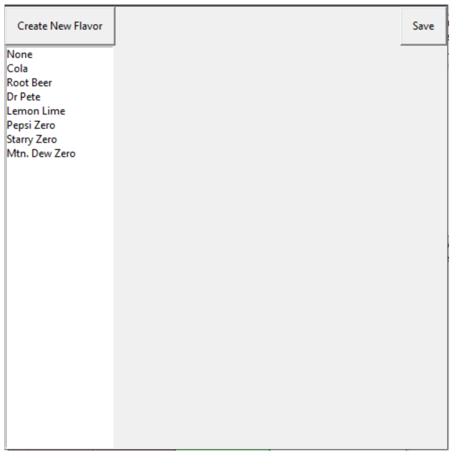
For the user to interface with the Soda Shop, they will interact with a touch screen programmed by a Raspberry Pi 4. The pi runs guizero in python to easily customize the user interface. The user can select from a programmed set of flavors to match the ones they have loaded into the machine. The user can then select a menu to view a set of soda flavors that can be made with their flavors. However, if the user would like to customize their drink, they can adjust sliders on each flavor to select how much of each option they want. The user can also change the name and color of the flavor that they input in case they are using a syrup flavor that hasn't been pre-programmed. Additionally, the user can set up to five favorite drinks. When customizing their drink, they will have the option to select a color associated with that drink. When displayed on the favorites section of the home screen, the drink will have the background of the selected color.

Once the user finalizes their drink, the Raspberry Pi will output five numbers through its USB serial port. A microcontroller will then read those numbers and output digital signals to the motors to pump out the requested amount of liquid. The microcontroller has a set total pump length of ten seconds. The percentages sent from the Raspberry Pi are a percent of that 10 total seconds. The sum of the percentages have to be either below 100 or at 100. This way, if the user wants 20% syrup from the first flavor and 60% carbonated water, the pumps will pump for two seconds and six seconds. This will ensure an accurate ratio.





Implementation



Prototyping Process

With a basic idea of selecting a button to automatically make a drink, a plan was set into motion to determine the best way to implement this. Each pump was planned to eventually be controlled via a fluid control board. This includes the flavor pumps, main carbonated water pump, any valves, and pressure sensors for safety. In its final form, only pumps were necessary to perform the desired operation.

To start, one peristaltic pump was purchased and powered from a bench power supply to check flow-rate, current, power consumption. With this information, it was determined the peristaltic pump was suitable, and three additional pumps were procured. After discussion on how to move the carbonated water, an air pump was purchased for testing as well. With the expectation that all pumps could run simultaneously, total expected current from pumps was calculated, then verified using the bench power supply directly, running all together.

To directly control the pumps, initially an on-board relay control system was designed, however this required producing multiple voltage levels to maintain dropout voltage of transistors and keep voltage high enough to actuate relays. As such, the group decided to use an external relay control board powered from 24 VDC directly. An external relay control board was procured and tested for acceptable operation.

Next the relay control board was controlled directly from a bench power supply to control the pumps, then from a microcontroller using a small test function, to test both the ability and compatibility using code written by Nic.

Testing of the flavor pumps determined flavor pumps could pump the recommended amount of flavor in 11 seconds for 16 oz of water. This is a reasonable amount of time, since a longer distribution time will allow for better flavor mixing using just the turbulence of the water. This removes the need for an external mixer for any created drinks. As the expected normal dispense volume is lower, the actual normal dispense time was also lower, especially when mixing flavors, as the total pumped volume of an individual flavor is reduced.

Functional Features

- The system can have up to four individual flavors, which can be dispensed independently or mixed for a broader range of flavor possibilities.

- The user will be able to select how much flavoring they want in their drink, giving more control for those who like their drinks stronger or weaker. There will be a default option for users who don't want to think about the amount of syrup, and just want their soda to taste similar to what you would expect from a can.
- Operation is controlled via a 7" touchscreen providing the user with complete control of ratios between water and each individual flavor.
- SodaShop allows a selection of pre-determined volumes at the press of a button.
- SodaShop is capable of storing hundreds of flavor combinations and five user-determined favorite recipes.
- Stored combinations automatically fall into valid and invalid recipe bins based on the entered current syrups installed to inform the user of selectable saved combinations based on current availability.
- Favorites can quickly be dispensed from the home screen to provide a drink with minimal wait time.
- The system can plug directly into a wall 120VAC outlet to provide fast and easy access and operation.
- SodaShop uses four peristaltic pumps for syrup control providing precise control and easy-to-swap tubing for replacement and cleaning.
- Main water is pumped via air pressure to prevent cavitation maximizing carbonation for the optimal drinking experience. This method also provide self-pressure relief preventing over-pressure conditions in the water container.
- The system is designed to allow any standard 2-liter bottle for water. Flavors are currently relegated to using third-party bottles with modified caps.

Testing and Validation

Fluid Flow: Initially, individual syrup pumps were powered to determine flow rate and current draw. Once these were known the main water air-pump was tested for its current draw and ability to move water in the manner intended. The air pressure was verified to provide adequate flow control of water. All pumps were then powered from a bench power supply to run for longer than momentary pulsing, and many times repeatedly, to provide a baseline for reliability. From here testing was halted until all pumps and relay-control boards were received.

A temporary control system using push buttons and an Arduino Nano was set up. Basic code was written to provide timed control of pumps at the press of a button. This provided management of initial flavor and water control. The code was also written to control all pumps at the press of a button to provide a basic drink configuration at worst case current conditions. This was used to verify both relay reliability as well as functional timing control.

During relay testing, it was found that the control boards being used were less than reliable such that two relays, both in the same position on their respective boards, another board was purchased for insurance purposes, however no other relays failed during testing up and into the product's presentation. For a commercial implementation, a custom relay control board with more reliable relays would be used to ensure long term continued proper operation.

Once the GUI was set up, the pumps were again tested to ensure timing of various pumps are the expected mixing ratios. Remarkably, the unit dispensed a viable soda mix on its first run, making additional timing and mixing adjustment fairly uneventful.

Challenges

While testing the relay control board, it was found that one relay on each board was inoperable, with three relays on each board functioning as expected. Since there are five motors and one solenoid being powered by the relays, this is still acceptable using two of the relay boards. As long as no other relays fail, this is not an issue.

It was found that with all motors and relays on the same bench power supply, the power supply would trip out on overcurrent. This was not an issue when connected directly to the 24VDC power supply, as the power supply has a higher current ability than the bench power supply.

Two challenges were overcome in the production of the custom printed circuit board. First the TVS diode array itself was shorting out following reflow. This appeared to be the solder paste used where it needed a higher temperature than the oven provided to flow properly. Unfortunately the TVS diode was destroyed while being removed. The PCB was able to run without the array, and extra care was taken going forward to prevent electrostatic discharge from damaging the board. The second challenge resulted from a missed connection on the 1.1 VDC control line of the RP2040 in the design. This prevented the RP2040 from powering on with power supplied. With the assistance of Dr. Bauer, the missed connection was found and a jumper added to the board to provide the necessary connection. Following the change, the PCB was functional and programmable. As may be expected, following reflow, there were a few other shorts in the system from excess solder which were cleaned up prior to attempting powering on the device initially.

The next challenge needing to be discussed is carbonation. A lot of people's favorite part of a soft drink other than the taste is the carbonation, so getting this part right for the project was a major objective and consideration. In the prototyping phase, we discovered that commercial carbonators cost around $800, and require massive amounts of additional equipment. So, in order to combat this we figured the user would get off best just buying club soda. However, this led to an additional issue: pumping carbonated water. After much deliberation and research, we figured using a rotary water pump would cause far too much turbulence which in turn would destroy the carbonation. The ultimate solution to this challenge was coming up with the idea of using differential pressure in order to pump the carbonated water. An air pump was utilized to offset the pressure of the 2 liter bottle, and when it turns on, it forces air into the bottle and thus liquid out of the second tube. This challenge was difficult to solve, but in the end the result worked better than expected, leaving a carbonated beverage to enjoy.

Additionally, there were many challenges involved with 3D printing and 3D design. First, as expected, many iterations were required for a majority of the parts for the project. These include: the drip tray, touch screen mount, T-nut mounts, syrup containers, electronic mounts, and more. Normally these were off by fractional amounts, due to error in precision of the 3D printers. We also needed to learn the properties of different filaments because a lot of them were breaking under any load. Overall, there weren't too many challenges for 3D printing that were unexpected, but definitely 3D printing and design took up a lot of time.

One challenge that arose during initial testing was the amount of splattering that would happen as syrup and water poured from the nozzle into the cup below. The nozzle had its bottom extended further down to reduce this problem. Some splattering still occurred, especially when the cup being filled was short, but the problem was reduced significantly.

During testing, we ran into the issue where the Raspberry Pi screen would dim automatically. When we pressed the screen to turn it back on, the dispense button got pressed, causing liquid to dispense all over with no cup. To fix this, a simple confirmation pop-up was added before dispensing. The Raspberry Pi's settings were changed to never dim also.

There were numerous bugs in the UI code that were found and fixed, but one bug persisted into the final design used at the showcase. If a user opened the "Create New Drink" menu, they could no longer edit an already made drink; pressing the "Edit Drink" button would instead bring them to the “Create New Drink” page. The menu logic between the two menus are similar, only a bool says which one to open. So, the bug most likely lies in that boolean not getting reset after opening the create drink page.

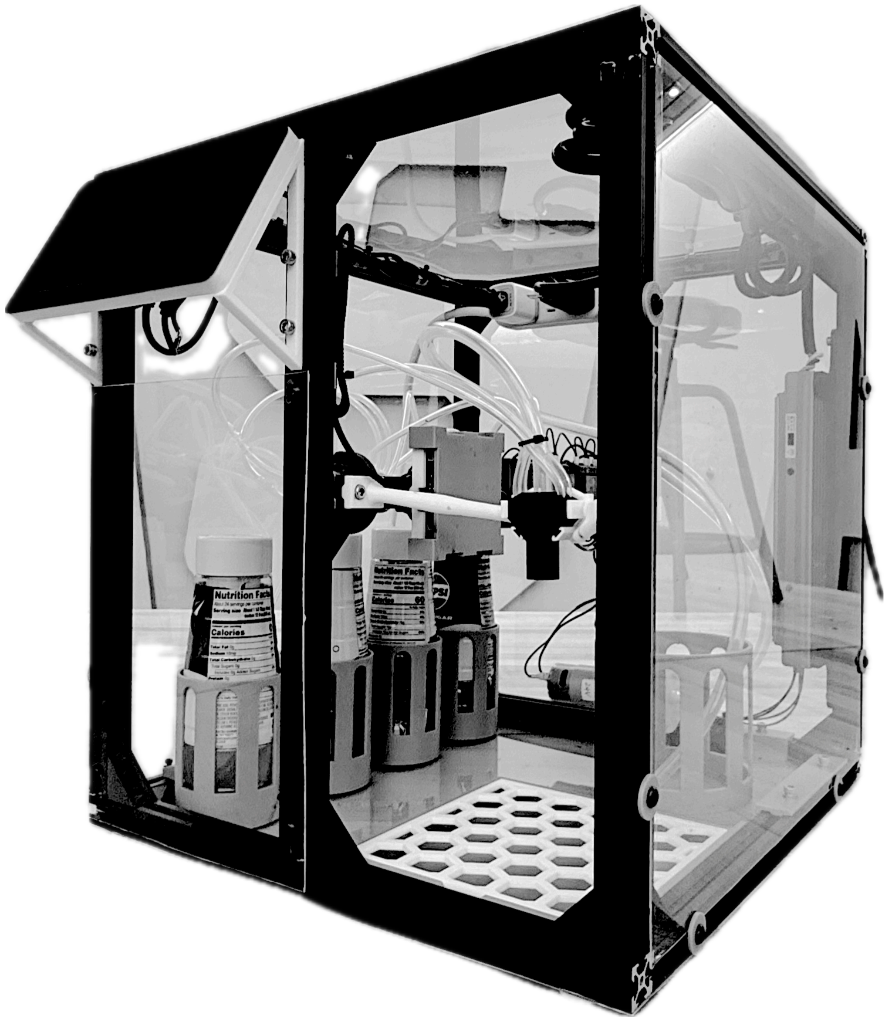
Results and Analysis

In the end, the SodaShop was able to pump 4oz of soda in 4 seconds. During this time, syrup pumps as well as our air pump would run for 2 seconds. The syrup would run precisely this amount of time while the carbonated water output would start flowing slightly later and would dribble out for approximately 2 seconds past the allotted time. Regardless, even with higher volumes, the amount of soda produced was directly and linearly correlated to the amount of time it took to produce.

The syrups are output at a rate of 1/3oz per 2 seconds. This amount is typically negligible in calculating final drink volume unless more than one syrup is run at a time. This also means approximately 44 drinks can be made from each syrup container. This means the final SodaShop product can produce 176 drinks on one set of syrups without having to be refilled. This fulfills the original volume for the plan.

On the other hand, only about 17 4oz drinks can be made from a 2L. This means that the seltzer water reservoir needs to be refilled much more frequently than the syrup. This is a much smaller amount than was originally desired. Further improvements in the future to mitigate this problem could involve adding a second 2L, connecting a water line and bypassing carbonation, or adding an expensive in-line carbonator. For the time being, our final product is in an acceptable state.

Discussion



Overall the project feels like a success. The system provides the user direct control to mix drinks using various concentrations of each flavor. This is accomplished via the GUI, touchscreen and Raspberry Pi. The system controls the five pumps to mix the drink to the set ratio. On top of this, the GUI allows for the storage, selection, editing, and otherwise control of recipes. Also, the list keeps track of recipes that are both valid and invalid and provides quick access to a select few favorite recipes. For all these reasons, the project is considered successful.

There were some limitations to the project. The original design had the inclusion of a carbonation system. This was found to be prohibitively expensive resulting in use of pre-carbonated water (seltzer water) for carbonated water for mixing. This also indicated, to provide a sub-100 dollar price point that a new carbonation system would need to be designed, though that is a mechanical engineering endeavor such that the time to produce a viable product preempted this.

Even with the success of the project, there is always room for improvement. The addition of a confirmation in the GUI prevents accidental dispensation. However, a future iteration could use a cup-detection method to only dispense when a receptacle is present under the nozzle. Additionally, adding inline carbonation to the system would be beneficial, however as previously discussed, this may increase the cost of the system substantially. Lastly, combining the microcontroller and relay control boards into a single board would allow for some consolidation and a single power point to the system reducing manufacturing costs.

Ethical Considerations

To take this product to mass production, ethical considerations need to be addressed across the lifespan of the machine. During component selection, care should be taken to ensure that electronics are sourced from suppliers that avoid unethical labor practices and use sustainable materials when possible. In addition to using large touch targets on the GUI, high-contrast display options could be added to increase accessibility.

While the unit is currently wholly offline, if it were to be internet connected, if data like usage statistics were to be collected, ethical handling of that data would be required including consent, anonymization, and secure storage protocols.

For the environment, the machine would need to be designed for energy efficiency and long-term use. Power draw from idle components like displays or valves would be measured and reduced where possible. All internal parts would be selected for ease of repair and replacement to extend service life. At end-of-life, recycling and safe disposal guidelines would be required, especially for components like batteries or electronic controllers. These considerations will be revisited if the machine were to move beyond prototype and into deployment at scale.

Health and Safety

In considering the health, safety, and welfare of users, three primary components were evaluated for direct impact on public health: the fluid tubing, dispensing nozzle, and drip tray.

The tubing, which carries either syrup and carbonated water, presents a risk of contamination if not properly maintained. Syrup in particular can be high in sugar content which can support growth of bacteria and fungi if left to sit long term. To address this, food-grade tubing was selected. Additionally, all tubing is user-replaceable. This decision was made to ensure that tubing showing signs of buildup or wear can be easily swapped out without requiring major disassembly of the machine. Replaceable tubing supports a maintenance model where hygiene can be preserved through regular part rotation rather than relying solely on cleaning protocols, reducing the likelihood of contamination and improving user safety. Though this introduces recurring user costs, it decreases the risk of user illness.

The dispensing nozzle was identified as a critical point in preventing cross-contamination. If poorly designed, a nozzle can allow splashback or buildup, introducing bacteria or allowing flavor carryover between drinks. To mitigate this, a removable and replaceable nozzle design was implemented. The nozzle assembly was designed to be easily detached without tools, allowing for regular manual cleaning as well as full replacement if contamination occurs. This also enables the use of spare or separate nozzles for different setups or during cleaning cycles, reducing downtime. Although this approach slightly increases part count and manufacturing complexity, it provides a safer and more hygienic user experience.

The drip tray below the nozzle and cup to be filled was evaluated for its impact on cleanliness and safety. Standing liquid in a poorly drained or fixed tray can lead to mold growth or overflow, posing both health and slipping hazards. To reduce these risks, a removable drip tray is incorporated into the design. The tray includes a sloped base and a slotted grate to encourage drainage, and the entire unit can be lifted out for cleaning. This allows users to regularly empty and sanitize the tray without specialized tools. While this adds to the mechanical design requirements, it also simplifies maintenance and reduces liquid buildup.

From an economic perspective, each of these components contributes to a balance between initial cost and long-term reliability. Replaceable tubing and nozzle assemblies increase consumable costs but reduce the likelihood of health-related issues. A removable drip tray reduces the need for professional cleaning or service due to spillage or mold. Altogether, these design choices prioritize user safety and hygiene while managing total cost of ownership. Each decision was made to improve long-term functionality and public health safety, even where it can result in some increase in part complexity or maintenance.

Standards

Given the electrical and mechanical nature of the system, several regulatory and safety standards apply. UL (Underwriters Laboratories), FCC (Federal Communications Commission), CE (Conformité Européenne), and ISO were reviewed against the electrical specifications and component layout of the prototype.

The power supplies used in the design, operating at 24 VDC and 5 VDC respectively, receive input from standard AC mains voltage (100–240 VAC) falling under UL 62368-1: Standard for Audio/Video, Information and Communication Technology Equipment – Part 1: Safety Requirements, which covers power supplies for information and electronic equipment. This standard includes custom embedded systems with displays and user interaction.

Electrical connections to the AC through the power supplies fall under UL 508: Standard for Industrial Control Equipment, as relays are housed in the enclosure. Additionally, power connectors and internal wiring will need to comply with UL 498: Standard for Safety of Attachment Plugs and Receptacles, to ensure that all high-voltage interfacing parts meet safety clearance, grounding, and insulation requirements in a production model.

Given the use of a touchscreen display and microcontroller, the system may also fall under IEC 61000-4-2 and IEC 61000-4-4, which define Electrostatic Discharge (ESD) and Electrical Fast Transients (EFT) immunity, respectively. These standards are part of CE certification for immunity performance and are relevant to ensure the machine remains functional and safe in environments with static discharge or electrical noise.

Fluid-handling components, including the peristaltic pumps and tubing, will need to conform to NSF/ANSI 18: Manual Food and Beverage Dispensing Equipment, in a production model. This governs materials and sanitary design for beverage dispensing machines. Tubing and nozzles that come into contact with consumable liquids must also comply with NSF/ANSI 51:

Food Equipment Materials to confirm that all wetted materials are non-toxic, cleanable, and suitable for food contact.

Finally, if this product were to be sold internationally, CE marking requirements would also apply. Under the Low Voltage Directive (LVD) 2014/35/EU and the EMC Directive 2014/30/EU, SodaShop would need to meet equivalent European standards for electrical safety and electromagnetic emissions/immunity.

Conclusion

Overall, the project worked as expected. The GUI provided the ability to program a myriad of flavor choices to control precise concentration ratios across one to four flavors and the main water. This created a consistent end product (soda) to the user through an easy to navigate menu structure. Some key achievements of the project included providing the four desired flavors, providing a pumping solution for the syrups as well as for the main water with a minimized decarbonation as the main water is pumped through application of air pressure only, and an easy to use interface to create, store, use, edit, delete, and select a multitude of recipes. Additionally, the GUI provides a list of valid and invalid recipes based on the current flavors connected ensuring the user doesn't accidentally try to make a drink that isn't possible at a given time. The addition of five favorites to the main screen also makes the machine fast and simple for everyday use!

The SodaShop provides an in-home solution to reduce plastic bottle usage, while providing the user complete control of their drink experience. A person can set their recipes to match their tastes specifically instead of being shoehorned into pre-purchased options. The use of peristaltic pumps with food-grade tubing provides a hygienic setup that can both be cleaned passing a solution, and by replacing the tubing periodically to mitigate any buildup or deterioration over time.

Miraculously, the system worked the first time everything was connected. The GUI communicated with the microcontroller and pumps without issue providing a viable product on the first go. The pumps were capable of providing the desired precise control setup using the initial testing volumetric findings. The pressure system used to pump main water worked flawlessly, and given the designed imperfect-seal connection alleviated the need for a pressure relief system as well. While there was some minor miscommunication between programming and pump control design, the system was able to be re-programmed with no negative impact on soda mixing quality. In general, the system worked well, however in testing, the relay control boards were found to be somewhat unreliable such that in a mass-produced version, the relays would be further investigated for better, more reliable options. Aside from two of the eight available failing out the gate, the remaining relays remained functional through all testing to the product demonstration, where nearly 16 liters of soda were produced over the course of around 3 hours of operation.

Some key achievements were completing the construction of the frame, working out the perfect syrup ratios for the best flavor using the available syrups, and cleaning up the user interface for better presentability. While the system functioned well from the start, the final product was far more refined and easier to navigate. The construction of the frame was a significant milestone, as the pumps and controller were ready to go but difficult to test without having a more permanent mounting system.

The SodaShop project is relevant because it provides a way to create soda or any drink using a mixing syrup and carbonated or flat water. It provides a measurable customization that isn't available in similar drink mixing systems, as well as removes the need for manual carbonization of water. The SodaShop project could easily be reworked and marketed as a household product, which was one of the goals of the project when it began. More significance is realized when SodaShop's ability to provide a specific, measurable amount of fluid is considered. This product could be easily adapted to provide specific amounts of liquid for nearly any application, not just for drinks. With this in mind, the potential of this project is nearly endless, especially in the world of drink mixing.

Some notable issues with the project were the seal of the water system, losing carbonation, and splashback from the nozzle dispenser. The issue of the water seal was discovered during the showcase presentation, when the volume of water was lessened from a loss of air in the pressurized pumping system. This was solved quite easily, by adjusting the hoses going through the cap into the water vessel to remove air gaps, with a noticeable increase in water flow afterwards. The loss of carbonation was discovered when trying to pump carbonated water through a peristaltic pump, which was solved by transitioning to the air pressure based system. By pumping air into a sealed vessel, it was an easy matter to push water through the tubing and into the nozzle, which greatly reduced the loss of carbonation. The issue of splashback was easily resolved by relocating the nozzle closer to the cup, and adjusting the depth of the syrup lines in the nozzle. This had the bonus effect of helping the syrup mix more thoroughly with the carbonated water, resulting in a better and more consistent flavor for each drink produced.

References

UL Solutions, “IEC 62368-1 Testing and Certification,” *UL*, [Online]. Available: https://www.ul.com/services/iec-62368-1-testing-certification. [Accessed: May 6, 2025].

UL Solutions, “Explore the Benefits of Industrial Control Panel Certification,” *UL*, [Online]. Available: https://www.ul.com/explore-benefits-industrial-control-panel-certification. [Accessed: May 6, 2025].

UL Solutions, “Attachment Plug and Receptacle Safety Evaluations,” *UL*, https://www.ul.com/services/attachment-plug-and-receptacle-safety-evaluations (accessed May 6, 2025).

NSF International, *NSF/ANSI 18-2022 – Manual Food and Beverage Dispensing Equipment*. Ann Arbor, MI: NSF International, 2022. [Online]. Available: https://webstore.ansi.org/standards/nsf/nsfansi182022. [Accessed: May 6, 2025].

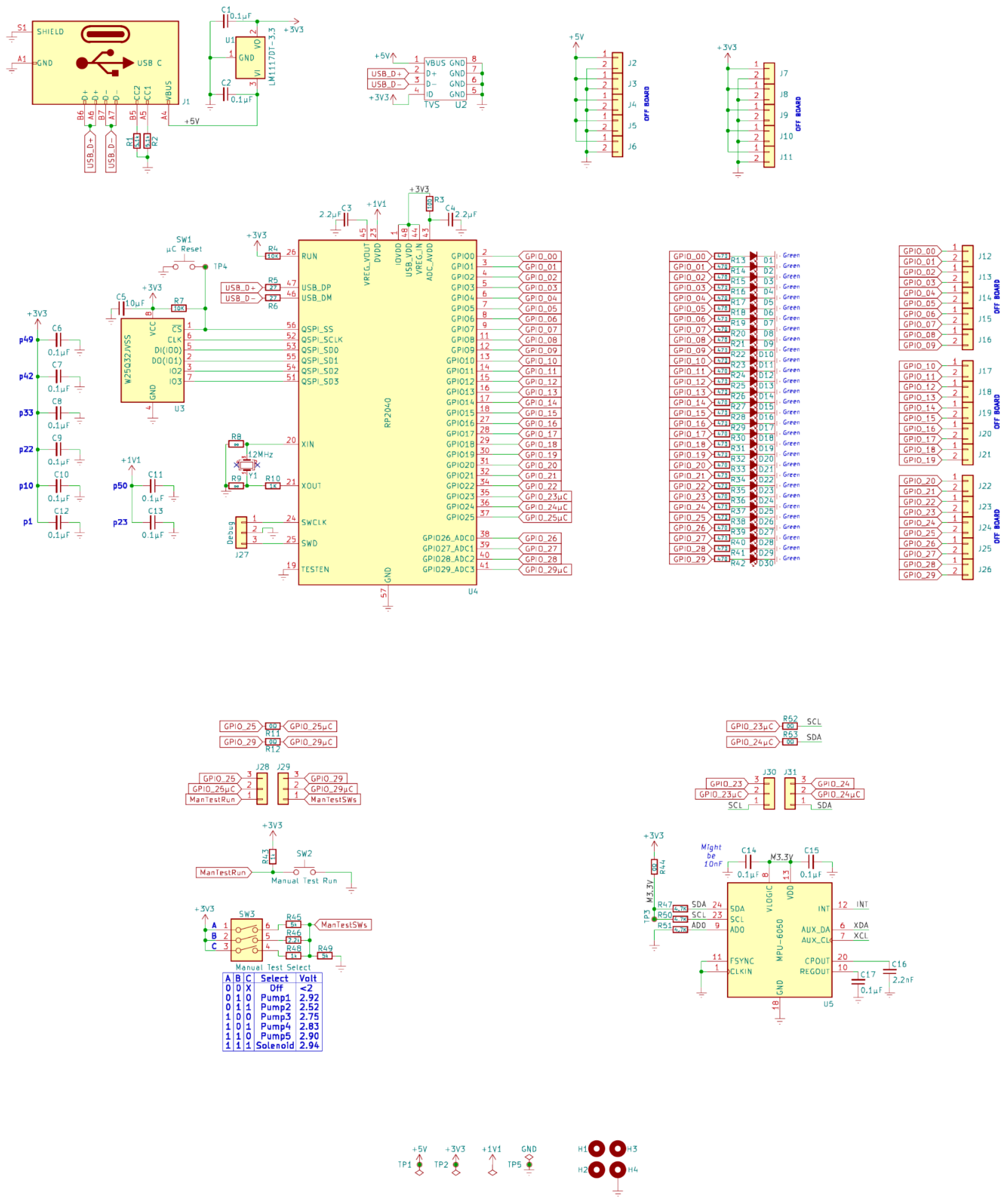
European Commission, “Electromagnetic Compatibility (EMC) Directive,” *Single Market Economy*, https://single-market-economy.ec.europa.eu/sectors/electrical-and-electronic-engineering-industries-eei/electromagnetic-compatibility-emc-directive_en (accessed May 6, 2025).

M. O’Hanlon and L. Sach, “guizero: A Python 3 library for creating simple GUIs,” *guizero Documentation*, https://lawsie.github.io/guizero/ (accessed May 6, 2025).

u/Neurodoc, “Pumping a carbonated liquid,” *Reddit*, r/AskEngineers, Jun. 28, 2014. [Online]. Available: https://www.reddit.com/r/AskEngineers/comments/290rbu/pumping_a_carbonated_liquid/

Appendices

Schematic Diagram of Microcontroller.



A	B	C	Select	Volt
0	0	X	Off	<2
0	1	0	Pump1	2.92
0	1	1	Pump2	2.52
1	0	0	Pump3	2.75
1	0	1	Pump4	2.83
1	1	0	Pump5	2.90
1	1	1	Solenoid	2.94

Budget and Cost Breakdown: Total $491.53

Item	Cost
Pepsi Syrups	12.00
4 Channel 24V Relay Module w/ Optocoupler High/Low Trigger x2	13.93
T Connectors	6.99
Peristaltic Pumps x4	39.92
Silicone Tubing, 1/4" ID x 3/8" OD High Temp Food Grade, 10 Feet	11.99
Air Pump	18.69
24V power supply	17.00
Aluminum Frame Rails	59.99
M5 Screw Kit	9.99
Plexiglass Sheets	33.29
Screw Hardware Kit	24.99
Cups	20.79
T-nut Set	12.99
Syrups x3	19.88
Syrups x4	24.95
Seltzer water x8	13.28
Custom designed PCB from OshPark	44.00
Silicone Tubing, 3mm ID x 5mm OD High Temp Food Grade, 3m	7.99
24V 400W Transformer PSU, AC/DC Converter, 100~265VAC Input	24.99
Raspberry Pi 4 (2GB)	58.91
Mini power strip	4.98
Assortment of 2.8mm, 4.8mm, and 6.3mm Female Spade Crimp Terminal with Insulating Sleeve and Wire, 25cm	9.99

Code snippets

main.py:

```
import parms
import homeMenu
import settings

def main():
  parms.init()
  settings.loadSettings()
  homeMenu.home()
  parms.app.display()

main()
```

parms.py:

```
import parms
import drinkMenu
import flavorsOutSerial
import flavorMenu
import favoriteMenu
import settings
from guizero import Box, ListBox, PushButton, Combo

# Select Drink Function
def selectDrink(selection):
  for drink in parms.drinks:
    if (selection == drink.name):
      parms.modify_value(parms.currentDrink, drink)
      break
  # Load Selected Drink
  if (not selection == None):
    if (parms.validDrinkSelection['value']):
      dispenseButton.text = "Dispense: " + parms.currentDrink['value'].name +
"\n"
    else:
      dispenseButton.text = "Not correct flavors, please load: "
      if (not parms.currentDrink['value'].flavor1Name in parms.chosenFlavors)
and (parms.currentDrink['value'].flavor1Perc > 0) and (not
parms.currentDrink['value'].flavor1Name == "None"):
        dispenseButton.text += parms.currentDrink['value'].flavor1Name + ", "
      if (not parms.currentDrink['value'].flavor2Name in parms.chosenFlavors)
and (parms.currentDrink['value'].flavor2Perc > 0) and (not
parms.currentDrink['value'].flavor2Name == "None"):
        dispenseButton.text += parms.currentDrink['value'].flavor2Name + ", "
      if (not parms.currentDrink['value'].flavor3Name in parms.chosenFlavors)
and (parms.currentDrink['value'].flavor3Perc > 0) and (not
parms.currentDrink['value'].flavor3Name == "None"):
        dispenseButton.text += parms.currentDrink['value'].flavor3Name + ", "
      if (not parms.currentDrink['value'].flavor4Name in parms.chosenFlavors)
and (parms.currentDrink['value'].flavor4Perc > 0) and (not
parms.currentDrink['value'].flavor4Name == "None"):
        dispenseButton.text += parms.currentDrink['value'].flavor4Name + ", "
      dispenseButton.text = dispenseButton.text[:-2] + "\n"

    dispenseButton.text += "Needed Flavors: "
    if (parms.currentDrink['value'].flavor1Perc > 0) and (not
parms.currentDrink['value'].flavor1Name == "None"):
      dispenseButton.text += parms.currentDrink['value'].flavor1Name + ", "
    if (parms.currentDrink['value'].flavor2Perc > 0) and (not
parms.currentDrink['value'].flavor2Name == "None"):
      dispenseButton.text += parms.currentDrink['value'].flavor2Name + ", "
    if (parms.currentDrink['value'].flavor3Perc > 0) and (not
parms.currentDrink['value'].flavor3Name == "None"):
      dispenseButton.text += parms.currentDrink['value'].flavor3Name + ", "
    if (parms.currentDrink['value'].flavor4Perc > 0) and (not
parms.currentDrink['value'].flavor4Name == "None"):
      dispenseButton.text += parms.currentDrink['value'].flavor4Name + ", "
    dispenseButton.text = dispenseButton.text[:-2]

def selectInvalidDrink(selection):
  parms.modify_value(parms.validDrinkSelection, False)
  selectDrink(selection)

def selectionValidDrink(selection):
  parms.modify_value(parms.validDrinkSelection, True)
  selectDrink(selection)

def createNewDrink():
    parms.modify_value(parms.newDrinkFlag, True)
    drinkMenu.editDrink()

def clearLists():
  invalidDrinkList.clear()
  validDrinkList.clear()
```

```
def updateDrinkNameLists():
  parms.validDrinks.clear()
  parms.invalidDrinks.clear()
  clearLists()
  chosenFlavorNames = []
  for chosenFlavor in parms.chosenFlavors:
    chosenFlavorNames.append(chosenFlavor.name)
  for drink in parms.drinks:
      if ((drink.flavor1Perc > 0) and (not (drink.flavor1Name in
chosenFlavorNames)) and (not drink.flavor1Name == "None")):
        parms.invalidDrinks.append(drink)
      elif ((drink.flavor2Perc > 0) and (not drink.flavor2Name in
chosenFlavorNames) and (not drink.flavor2Name == "None")):
        parms.invalidDrinks.append(drink)
      elif ((drink.flavor3Perc > 0) and (not drink.flavor3Name in
chosenFlavorNames) and (not drink.flavor3Name == "None")):
        parms.invalidDrinks.append(drink)
      elif ((drink.flavor4Perc > 0) and (not drink.flavor4Name in
chosenFlavorNames) and (not drink.flavor4Name == "None")):
        parms.invalidDrinks.append(drink)
      else:
        parms.validDrinks.append(drink)
  for validDrink in parms.validDrinks:
    validDrinkList.append(validDrink.name)
  for invalidDrink in parms.invalidDrinks:
    invalidDrinkList.append(invalidDrink.name)

def setFlavorColor1():
  parms.chosenFlavors[0] = parms.Flavor(flavor1Button.value,
parms.findFlavorFromName(flavor1Button.value).color)
  flavor1Button.bg = parms.chosenFlavors[0].color
  updateDrinkNameLists()
  settings.saveSettings()
def setFlavorColor2():
  parms.chosenFlavors[1] = parms.findFlavorFromName(flavor2Button.value)
  flavor2Button.bg = parms.chosenFlavors[1].color
  updateDrinkNameLists()
  settings.saveSettings()
def setFlavorColor3():
  parms.chosenFlavors[2] = parms.findFlavorFromName(flavor3Button.value)
  flavor3Button.bg = parms.chosenFlavors[2].color
  updateDrinkNameLists()
  settings.saveSettings()
def setFlavorColor4():
  parms.chosenFlavors[3] = parms.findFlavorFromName(flavor4Button.value)
  flavor4Button.bg = parms.chosenFlavors[3].color
  updateDrinkNameLists()
  settings.saveSettings()

def updateFlavorColors():
  setFlavorColor1()
  setFlavorColor2()
  setFlavorColor3()
  setFlavorColor4()

def updateFlavorButtons():
  flavor1Button.clear()
  flavor2Button.clear()
  flavor3Button.clear()
  flavor4Button.clear()
  for flavor in parms.flavors:
    flavor1Button.append(flavor.name)
    flavor2Button.append(flavor.name)
    flavor3Button.append(flavor.name)
    flavor4Button.append(flavor.name)
  flavor1Button.value = parms.chosenFlavors[0].name
  flavor1Button.bg = parms.chosenFlavors[0].color
  flavor2Button.value = parms.chosenFlavors[1].name
  flavor2Button.bg = parms.chosenFlavors[1].color
  flavor3Button.value = parms.chosenFlavors[2].name
  flavor3Button.bg = parms.chosenFlavors[2].color
  flavor4Button.value = parms.chosenFlavors[3].name
  flavor4Button.bg = parms.chosenFlavors[3].color

def updateFavoriteButtons():
  for i in range(len(parms.favoriteDrinks)):
    favoriteDrinkButtons[i].bg = parms.favoriteDrinks[i].color
    favoriteDrinkButtons[i].text = parms.favoriteDrinks[i].name

def checkFavoriteValid(selection):
  drink = parms.favoriteDrinks[int(selection)]
  chosenFlavorNames = parms.getListOfChosenFlavorNames()
  if ((not(drink.flavor1Name in chosenFlavorNames)) and (not (drink.flavor1Name
== "None")) and (drink.flavor1Perc > 0)):
    warning = parms.app.warn("Warning", "Please load the drink's correct
flavors")
  elif ((not(drink.flavor2Name in chosenFlavorNames)) and (not
(drink.flavor2Name == "None")) and (drink.flavor2Perc > 0)):
    warning = parms.app.warn("Warning", "Please load the drink's correct
flavors")
  elif ((not(drink.flavor3Name in chosenFlavorNames)) and (not
(drink.flavor3Name == "None")) and (drink.flavor3Perc > 0)):
    warning = parms.app.warn("Warning", "Please load the drink's correct
flavors")
  elif ((not(drink.flavor4Name in chosenFlavorNames)) and (not
(drink.flavor4Name == "None")) and (drink.flavor4Perc > 0)):
    warning = parms.app.warn("Warning", "Please load the drink's correct
flavors")
  else:
    flavorsOutSerial.sendFlavorValues(selection)

def home():
    # Make widgets global
    global validDrinkList
    global invalidDrinkList
    global flavor1Button
    global flavor2Button
    global flavor3Button
    global flavor4Button
    global dispenseButton
    global favoriteDrinkButtons

    # Main menu logic

    drinkListBox = Box(parms.app, height="fill", align="right")
    validDrinkList = ListBox(drinkListBox,
items=parms.getListOfValidDrinksNames(), height="fill", align="top",
command=selectionValidDrink, scrollbar=True)
    invalidDrinkList = ListBox(drinkListBox,
items=parms.getListOfInvalidDrinksNames(), height="fill", align="bottom",
command=selectInvalidDrink, scrollbar = True)
    invalidDrinkList.bg = "#e0dcdc"

    updateDrinkNameLists()

    flavorsSettingsBox = Box(parms.app, width="fill", align="bottom",
border=True)
    flavor1Button = Combo(flavorsSettingsBox,
options=parms.getListOfFlavorNames(), selected=parms.chosenFlavors[0].name,
align="left", command=setFlavorColor1)
    flavor1Button.bg = parms.chosenFlavors[0].color
    flavor2Button = Combo(flavorsSettingsBox,
options=parms.getListOfFlavorNames(), selected=parms.chosenFlavors[1].name,
align="left", command=setFlavorColor2)
```

```
    flavor2Button.bg = parms.chosenFlavors[1].color
    flavor3Button = Combo(flavorsSettingsBox,
options=parms.getListOfFlavorNames(), selected=parms.chosenFlavors[2].name,
align="left", command=setFlavorColor3)
    flavor3Button.bg = parms.chosenFlavors[2].color
    flavor4Button = Combo(flavorsSettingsBox,
options=parms.getListOfFlavorNames(), selected=parms.chosenFlavors[3].name,
align="left", command=setFlavorColor4)
    flavor4Button.bg = parms.chosenFlavors[3].color

    settingsBox = Box(parms.app, width = "fill", align="top", border=True)

    editFlavors = PushButton(settingsBox, text="Edit Flavors", align="left",
command=flavorMenu.editFlavor)
    editFavoriteDrinks = PushButton(settingsBox, text="Edit Favorite Drinks",
align="left", command=favoriteMenu.editFavorites)
    editDrinkButton = PushButton(settingsBox, text="Edit Selected Drink",
align="left", command=drinkMenu.editDrink)
    createNewDrinkButton = PushButton(settingsBox, text="Create New Drink",
align="left", command=createNewDrink)

    favoriteDrinkButtons = []
    for i in range(len(parms.favoriteDrinks)):
      favoriteButton = PushButton(parms.app, text=parms.favoriteDrinks[i].name,
width="fill", height="fill", command=checkFavoriteValid, args=str(i))
      favoriteButton.bg = parms.favoriteDrinks[i].color
      favoriteDrinkButtons.append(favoriteButton)

    dispenseButton = PushButton(parms.app, text="Select a Drink", width="fill",
height="fill", command=flavorsOutSerial.sendFlavorValues, args=["custom"])
```

flavorMenu.py:

```
import parms
import settings
import homeMenu
import subprocess
import os
from guizero import Window, Box, PushButton, ListBox, TextBox

# Edit Flavor Menu
def editFlavor():
  def show_keyboard(event=None):
    env = os.environ.copy()
    env["DISPLAY"] = ":0"
    subprocess.Popen(["onboard"],env=env)

  def hide_keyboard(event=None):
    subprocess.Popen(["pkill","onboard"])

  def on_return(event=None):
    hide_keyboard()

  def saveAndCloseWindow():
    settings.saveSettings()
    homeMenu.updateDrinkNameLists()
    homeMenu.updateFlavorButtons()
    closeWindow()

  def closeWindow():
    flavorWindow.hide()
    flavorWindow.destroy()

  def createNewFlavor():
    show_keyboard()
    flavorQuestion = flavorWindow.question("Flavor Edit", "Enter new flavor
name")
    hide_keyboard()
    if (not flavorQuestion in parms.flavors):
      parms.flavors.append(parms.Flavor(flavorQuestion, "#ffffff"))
      flavorsList.append(flavorQuestion)
      settings.saveSettings()

  def selectFlavor(selection):
    currentFlavor = selection
    chooseFlavor(True)

  def chooseFlavor(show):
    colorButton.show()
    colorButton.bg = next((flavor.color for flavor in parms.flavors if
flavor.name == flavorsList.value), "#ffffff")
    saveButton.show()
    copyFlavorButton.show()
    deleteButton.show()
    if (show):
      colorButton.show()
      colorButton.bg = next((flavor.color for flavor in parms.flavors if
flavor.name == flavorsList.value), "#ffffff")
      copyFlavorButton.show()
      deleteButton.show()
    else:
      colorButton.hide()
      copyFlavorButton.hide()
      deleteButton.hide()

  def selectColor():
    color = "None"
    color = flavorWindow.select_color(color=next((flavor.color for flavor in
parms.flavors if flavor.name == flavorsList.value), "#ffffff"))
    if (color == None):
      color = "#ffffff"
    for i, flavor in enumerate(parms.flavors):
      if flavor.name == flavorsList.value:
        parms.flavors[i].color = color
    colorButton.bg = color

  def copyFlavor():
    selectedFlavor = parms.findFlavorFromName(flavorsList.value)
    parms.flavors.append(selectedFlavor)
    updateFlavorList()

  def deleteFlavor():
    for i, flavor in enumerate(parms.flavors):
      if parms.flavors[i].name == flavorsList.value:
        parms.flavors.pop(i)
        chooseFlavor(False)
        break
    for i, chosenFlavor in enumerate(parms.chosenFlavors):
      if parms.chosenFlavors[i].name == flavorsList.value:
        parms.chosenFlavors[i] = parms.Flavor("None", "#ffffff")
    updateFlavorList()

  def updateFlavorList():
    flavorsList.clear()
    for flavor in parms.flavors:
      flavorsList.append(flavor.name)

  flavorWindow = Window(parms.app, title="Edit Flavors")
  flavorWindow.show(wait=True)
  flavorWindow.set_full_screen()
  settingsBox = Box(flavorWindow, width="fill", align="top")
  saveButton = PushButton(settingsBox, text="Save", command=saveAndCloseWindow,
align="right")
  newFlavorButton = PushButton(settingsBox, text="Create New Flavor",
command=createNewFlavor, align="left")
  deleteButton = PushButton(settingsBox, text="Delete", align="left",
command=deleteFlavor)
  deleteButton.hide()
  copyFlavorButton = PushButton(settingsBox, text="Copy Selected Flavor",
command=copyFlavor, align="left")
  copyFlavorButton.hide()
  flavorsList = ListBox(flavorWindow, items=[flavor.name for flavor in
parms.flavors], align="left", height="fill", command=chooseFlavor)
  colorButton = PushButton(flavorWindow, text="Color", width="fill",
command=selectColor)
  colorButton.hide()
  currentFlavor = "None"
```

favoriteMenu.py:

```
import parms
import settings
import homeMenu
from guizero import Window, Box, PushButton, ListBox

def editFavorites():
  def saveAndCloseWindow():
    import homeMenu
    homeMenu.updateFavoriteButtons()
    favoriteEditWindow.hide()
    favoriteEditWindow.destroy()

  def selectFavoriteDrink(selection):
    parms.modify_value(selectedDrink, int(selection))
    for i in range(len(parms.favoriteDrinks)):
      favoriteDrinkButtons[i].bg = parms.favoriteDrinks[i].color
    favoriteDrinkButtons[selectedDrink['value']].bg = "#e0dcdc"

  def chooseDrink(selection):
    for drink in parms.drinks:
      if selection == drink.name:
        parms.favoriteDrinks[selectedDrink['value']] = drink
    refreshFavoriteButtons()

  def refreshFavoriteButtons():
    for i in range(len(parms.favoriteDrinks)):
      favoriteDrinkButtons[i].text = parms.favoriteDrinks[i].name
      favoriteDrinkButtons[i].bg = parms.favoriteDrinks[i].color
    favoriteDrinkButtons[selectedDrink['value']].bg = "#e0dcdc"

  favoriteEditWindow = Window(parms.app, title="Edit Favorite Drinks")
  favoriteEditWindow.show(wait=True)
  favoriteEditWindow.set_full_screen()
  settingsBox = Box(favoriteEditWindow, width = "fill", align="top")
  exitButton = PushButton(settingsBox, text="Close and Save",
command=saveAndCloseWindow, align="left")

  selectedDrink = {'value':0}
  drinkList = ListBox(favoriteEditWindow, items=parms.getListOfDrinkNames(),
align="right", height="fill", command=chooseDrink, scrollbar=True)
  favoriteDrinkButtons = []
  for i in range(len(parms.favoriteDrinks)):
    favoriteButton = PushButton(favoriteEditWindow,
text=parms.favoriteDrinks[i].name, width="fill", height="fill",
command=selectFavoriteDrink, args=str(i))
    favoriteButton.bg = parms.favoriteDrinks[i].color
    favoriteDrinkButtons.append(favoriteButton)
  favoriteDrinkButtons[0].bg = "#e0dcdc"
```

drinkMenu.py:

```
import parms
import settings
import subprocess
import os
import time
import threading
from guizero import Slider, Combo, Window, PushButton, Box, Text, TextBox

def show_keyboard(event=None):
  env = os.environ.copy()
  env["DISPLAY"] = ":0"
  subprocess.Popen(["onboard"],env=env)

def hide_keyboard(event=None):
  subprocess.Popen(["pkill","onboard"])

def on_return(event=None):
  hide_keyboard()

# Edit Selected Drink Menu
def editDrink():
  drink = parms.currentDrink['value']
  if (parms.newDrinkFlag['value']):
    drink = parms.Drink("Enter Drink Name", 0, 0, 0, 0, 0,
parms.chosenFlavors[0], parms.chosenFlavors[1], parms.chosenFlavors[2],
parms.chosenFlavors[3], "#ffffff")

  # When Exit Button Pressed, Close New Drink Window
  def closeWindow():
    # TODO: Uncomment when on Pi
    hide_keyboard()
    makeDrinkWindow.hide()
    makeDrinkWindow.destroy()

  def editSliderConstraints(slider):
    """Adjusts slider constraints dynamically while ignoring 'None' sliders and
hiding unused ones."""
    active_sliders = [s for l, s in zip(flavor_labels, sliders) if l.value !=
"None"]
    active_sliders.append(carbSlider)  # Always include Carbonation

    total = sum(s.value for s in active_sliders)
    remaining = max(100 - total, 0)

    for s in active_sliders:
      if s != slider:
        s.end = min(100, s.value + remaining)

      # Hide sliders with 0 value if total is 100
      if total == 100 and s.value == 0:
        s.width = 0  # Hide slider
      else:
        min_width = 50
        s.width = max(min_width, int((s.end / 100) * parms.max_slider_width))

  def saveDrink():
    import homeMenu
    homeMenu.dispenseButton.text = "Select a Drink"
    newDrink = parms.Drink(nameText.value, flavor1Slider.value,
flavor2Slider.value, flavor3Slider.value, flavor4Slider.value,
carbSlider.value,
                     flavor1Label.value, flavor2Label.value,
flavor3Label.value, flavor4Label.value, colorButton.bg)
    parms.drinks.append(newDrink)

    if (not parms.newDrinkFlag['value']):
      parms.drinks.remove(drink)
      if (parms.validDrinkSelection['value']):
        homeMenu.validDrinkList.remove(drink.name)
      else:
        homeMenu.invalidDrinkList.remove(drink.name)
      if drink.name in parms.getListOfFavoriteNames():
        for i in range(len(parms.favoriteDrinks)):
          if parms.favoriteDrinks[i].name == drink.name:
            parms.favoriteDrinks[i] = newDrink

    homeMenu.updateFavoriteButtons()
    homeMenu.updateDrinkNameLists()
    parms.modify_value(parms.newDrinkFlag, False)
    settings.saveSettings()
    closeWindow()

  def updateSliders():
    """Updates sliders when a Combo box changes."""
    for label, slider in zip(flavor_labels, sliders):
        if label.value == "None":
            slider.value = 0  # Set slider to 0 if None is selected
            slider.width = 0  # Hide the slider
        else:
            slider.width = parms.max_slider_width  # Restore width for active
sliders

    editSliderConstraints(None)  # Recalculate constraints

  def deleteDrink():
    import homeMenu
    parms.drinks.remove(drink)
    if (parms.validDrinkSelection['value']):
      homeMenu.validDrinkList.remove(drink.name)
    else:
      homeMenu.invalidDrinkList.remove(drink.name)
    settings.saveSettings()
    homeMenu.dispenseButton.text = "Select a Drink"
    closeWindow()

  def changeColor():
    color = "None"
    color = makeDrinkWindow.select_color(color=drink.color)
    if (color == None):
      color = "#ffffff"
    colorButton.bg = color

  def selectFlavorCombo1():
    flavor1Label.bg = parms.findFlavorFromName(flavor1Label.value).color
    updateSliders()
  def selectFlavorCombo2():
    flavor2Label.bg = parms.findFlavorFromName(flavor2Label.value).color
    updateSliders()
  def selectFlavorCombo3():
    flavor3Label.bg = parms.findFlavorFromName(flavor3Label.value).color
    updateSliders()
  def selectFlavorCombo4():
    flavor4Label.bg = parms.findFlavorFromName(flavor4Label.value).color
    updateSliders()

  makeDrinkWindow = Window(parms.app, title="Create your drink")
  makeDrinkWindow.show(wait=True)
  makeDrinkWindow.set_full_screen()
  settingsBox = Box(makeDrinkWindow, width="fill", align="top")

  exitButton = PushButton(settingsBox, text="Back", command=closeWindow,
align="left")

  colorButton = PushButton(makeDrinkWindow, text="Color", command=changeColor,
width="fill", align="top")
  colorButton.bg = drink.color

  flavorEditterBox = Box(makeDrinkWindow, layout="grid", align="left")

  displayedFlavors = []
  if (parms.newDrinkFlag['value']):
    displayedFlavors.append(parms.chosenFlavors[0].name)
    displayedFlavors.append(parms.chosenFlavors[1].name)
    displayedFlavors.append(parms.chosenFlavors[2].name)
    displayedFlavors.append(parms.chosenFlavors[3].name)
  else:
    displayedFlavors.append(drink.flavor1Name)
    displayedFlavors.append(drink.flavor2Name)
    displayedFlavors.append(drink.flavor3Name)
    displayedFlavors.append(drink.flavor4Name)

  flavor1Label = Combo(flavorEditterBox, grid=[0,0],
options=parms.getListOfFlavorNames(), command=selectFlavorCombo1)
  flavor1Label.value = displayedFlavors[0]
  flavor1Slider = Slider(flavorEditterBox, grid=[1,0],
command=editSliderConstraints, align="left", height="20")
  flavor1Slider.value = drink.flavor1Perc

  flavor2Label = Combo(flavorEditterBox, options=parms.getListOfFlavorNames(),
grid=[0,1], command=selectFlavorCombo2)
  flavor2Label.value = displayedFlavors[1]
  flavor2Slider = Slider(flavorEditterBox, grid=[1, 1],
command=editSliderConstraints, align="left", height="20")
  flavor2Slider.value = drink.flavor2Perc

  flavor3Label = Combo(flavorEditterBox, options=parms.getListOfFlavorNames(),
grid=[0,2], command=selectFlavorCombo3)
  flavor3Label.value = displayedFlavors[2]
  flavor3Slider = Slider(flavorEditterBox, grid=[1,2],
command=editSliderConstraints, align="left", height="20")
  flavor3Slider.value = drink.flavor3Perc

  flavor4Label = Combo(flavorEditterBox, options=parms.getListOfFlavorNames(),
grid=[0,3], command=selectFlavorCombo4)
  flavor4Label.value = displayedFlavors[3]
  flavor4Slider = Slider(flavorEditterBox, grid=[1,3],
command=editSliderConstraints, align="left", height="20")
  flavor4Slider.value = drink.flavor4Perc

  carbLabel = Text(flavorEditterBox, text="Carbonation", grid=[0,4],
enabled=False)
  carbSlider = Slider(flavorEditterBox, grid=[1,4],
command=editSliderConstraints, align="left", height="20")
  carbSlider.value = drink.carbPerc

  flavor_labels = [flavor1Label, flavor2Label, flavor3Label, flavor4Label]
  sliders = [flavor1Slider, flavor2Slider, flavor3Slider, flavor4Slider]

  updateSliders()

  blur_zone = Box(makeDrinkWindow, width="fill", height="fill")
  blur_zone.tk.bind("<Button-1>", lambda e: makeDrinkWindow.tk.focus())

  saveButton = PushButton(settingsBox, text="Save", align="right",
command=saveDrink)
  deleteButton = PushButton(settingsBox, text="Delete", align="left",
command=deleteDrink)
```

```
  nameText = TextBox(settingsBox, text=drink.name, align="top", width="fill")

  # TODO: Uncomment when on Pi
  nameText.tk.bind("<FocusIn>", show_keyboard)
  nameText.tk.bind("<FocusOut>", hide_keyboard)
  nameText.tk.bind("<Return>", on_return)
```

flavorsOutSerial.py:

```
import parms
import serial
from guizero import App

def sendFlavorValues(selection):
  dispensingDrink = parms.currentDrink['value']
  if selection != "custom":
    dispensingDrink = parms.favoriteDrinks[int(selection)]

  popup = parms.app.yesno("Confirm Dispense", "Are you sure you want to
dispense " + dispensingDrink.name + "?")
  if (popup == False):
    return

  p = [0, 0, 0, 0, dispensingDrink.carbPerc]

  dispensingDrinkFlavors = [dispensingDrink.flavor1Name,
dispensingDrink.flavor2Name, dispensingDrink.flavor3Name,
dispensingDrink.flavor4Name]

  chosenFlavorNames = parms.getListOfChosenFlavorNames()
  for i, drinkFlavor in enumerate(dispensingDrinkFlavors):
    for j, selectedFlavor in enumerate(chosenFlavorNames):
      if drinkFlavor == selectedFlavor:
        print (drinkFlavor)
        print (selectedFlavor)
        print (dispensingDrink.flavor1Perc)
        print(i, j)
        if j == 0:
          if (i == 0):
            p[0] = dispensingDrink.flavor1Perc
          elif (i == 1):
            p[0] = dispensingDrink.flavor2Perc
          elif (i == 2):
            p[0] = dispensingDrink.flavor3Perc
          elif (i == 3):
            p[0] = dispensingDrink.flavor4Perc
        elif j == 1:
          if (i == 0):
            p[1] = dispensingDrink.flavor1Perc
          elif (i == 1):
            p[1] = dispensingDrink.flavor2Perc
          elif (i == 2):
            p[1] = dispensingDrink.flavor3Perc
          elif (i == 3):
            p[1] = dispensingDrink.flavor4Perc
        elif j == 2:
          if (i == 0):
            p[2] = dispensingDrink.flavor1Perc
          elif (i == 1):
            p[2] = dispensingDrink.flavor2Perc
          elif (i == 2):
            p[2] = dispensingDrink.flavor3Perc
          elif (i == 3):
            p[2] = dispensingDrink.flavor4Perc
        elif j == 3:
          if (i == 0):
            p[3] = dispensingDrink.flavor1Perc
          elif (i == 1):
            p[3] = dispensingDrink.flavor2Perc
          elif (i == 2):
            p[3] = dispensingDrink.flavor3Perc
          elif (i == 3):
            p[3] = dispensingDrink.flavor4Perc

print(p[0], p[1], p[2], p[3], p[4])

# TODO: Uncomment when on Pi
SERIAL_PORT = '/dev/ttyACM0'
BAUD_RATE = 115200

try:
  with serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1) as ser:
    data = f"{p[0]},{p[1]},{p[2]},{p[3]},{p[4]}\n"
    ser.write(data.encode('utf-8'))
    print(f"Sent: {data.strip()}")
except serial.SerialException as e:
  print("Serial error:", e)
```

settings.txt:

```
Chosen Flavors
None,#ffffff
Pepsi Zero,#7b5144
Root Beer,#764949
Lemon Lime,#80ff80
All Flavors
None,#ffffff
Cola,#804040
Root Beer,#764949
Dr Pete,#800000
Lemon Lime,#80ff80
Pepsi Zero,#7b5144
Starry Zero,#defc83
Mtn. Dew Zero,#04fb29
Favorite Drinks
Pepsi Zero,20,0,0,0,20,Pepsi Zero,None,None,None,#804040
Starry Zero,20,0,0,0,20,Starry Zero,None,None,None,#13ec80
Mtn. Dew Zero,20,0,0,0,20,Mtn. Dew Zero,None,None,None,#47e61a
Lemon Lime,20,0,0,0,20,Lemon Lime,None,None,None,#80ff80
Root Beer,20,0,0,0,20,Root Beer,None,None,None,#783d07
All Drinks
Pepsi Zero,20,0,0,0,20,Pepsi Zero,None,None,None,#804040
Mtn. Dew Zero,20,0,0,0,20,Mtn. Dew Zero,None,None,None,#47e61a
Starry Zero,20,0,0,0,20,Starry Zero,None,None,None,#13ec80
Root Beer,20,0,0,0,20,Root Beer,None,None,None,#783d07
Dr Pete,20,0,0,0,20,Dr Pete,None,None,None,#895136
Cola,20,0,0,0,20,Cola,None,None,None,#804040
Lemon Lime,20,0,0,0,20,Lemon Lime,None,None,None,#80ff80
Suicide,5,5,5,5,20,Root Beer,Pepsi Zero,Starry Zero,Lemon
Lime,#804000
Enter Drink Name,0,0,0,0,0,None,Pepsi Zero,Root Beer,Lemon
Lime,#ffffff
```